Escola de Camins
Escola Tècnica Superior d'Enginyeria de Camins, Canals i Ports
UPC BARCELONATECH

TREBALL FINAL DE GRAU

Desarrollo de una interfaz para la imposición de condiciones de contorno variables para un código de elementos finitos

Treball realitzat per:

Marc Busquets Casals

Dirigit per:
Antonia Larese, Pooyan Dadvan i Massimo Petracca

Grau en:
Enginyeria de la Construcció

Barcelona, juny de 2014

Departament de Resistència de Materials i Estructures a l'Enginyeria (RMEE)

DESARROLLO DE UNA INTERFAZ PARA LA IMPOSICIÓN DE CONDICIONES DE CONTORNO VARIABLES PARA UN CÓDIGO DE ELEMENTOS FINITOS

Autor: Marc Busquets Casals

Tutores: Antonia Larese, Pooyan Dadvan, Massimo Petracca

RESUMEN

Palabras claves: *condiciones de contorno variables, interfaz, acciones dinámicas, método de los elementos finitos, Kratos, CIMNE, Python.*

En el análisis estructural es muy importante y determinante estudiar cualquiera de las acciones, ya sean acciones estáticas o **acciones dinámicas**. Con la aparición del **método de los elementos finitos** y del cálculo por ordenador se ha permitido mejorar enormemente la capacidad de análisis de los problemas.

Kratos, una herramienta desarrollada por **CIMNE**, permite calcular mediante el método de los elementos finitos problemas de análisis de estructuras juntamente con otros de múltiples disciplinas de la ingeniería y de la ciencia. A través de sus aplicaciones permite implementar una gran variedad de condiciones de contorno, pero hasta el momento sólo existe la posibilidad de aplicarlas mediante un determinado valor constante.

En este trabajo se pretende desarrollar una **interfaz** para Kratos, con el lenguaje de programación **Python**, que posibilite la aplicación de **condiciones de contorno variables** en el tiempo de forma que se puedan analizar problemas sometidos a acciones dinámicas. Esta interfaz debe seguir la idea de fondo de Kratos, en el sentido de ser adaptable a una gran variedad de campos de estudio, no simplemente focalizando la aplicación en el análisis estructural. Para llevar a cabo la validación, se procederá a la realización de algunos ejemplos, comparando los resultados obtenidos con los teóricos y los de otros programas que usan también el método de los elementos finitos.

DEVELOPMENT OF AN INTERFACE FOR THE IMPOSITION OF BOUNDARY CONTIDITONS FOR VARIABLES OF FINITE ELEMENT CODE

Autor: Marc Busquets Casals

Tutores: Antonia Larese De Tetto, Pooyan Dadvan, Massimo Petracca

ABSTRACT

Key words: *variable boundary conditions, interface, dynamic actions, finite element method, Kratos, CIMNE, Python.*

Structural analysis is very important and crucial to study static or **dynamic actions**. Recently, as a result of the appearance of the **finite element method** and the improvement of computer computation, the capacity to analyze problems has increased significantly.

Kratos, a tool developed by **CIMNE**, allows solving structural analysis problems among other multiple disciplines of engineering and science using the finite element method. A great variety of boundary conditions can be imposed through its applications nonetheless at the moment there is only the possibility to apply it as a constant boundary conditions.

This work aims to develop an **interface** for Kratos, with the **Python** programming language, which enables imposing **variable boundary conditions** over time, so problems can be analysed under dynamic actions. This interface should follow the basic idea of Kratos in the sense of being adaptable to a variety of fields, not just focusing on the structural problem. To perform the validation some examples are created and run comparing the results obtained with the theoretical and with other programs that also use the finite element method.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

INTRODUCCIÓN

1.1 INTRODUCCIÓN

El análisis estructural es un tipo de proceso sistemático que trata de analizar el comportamiento de una estructura bajo el efecto de unas determinadas acciones exteriores, las cuales producen unos efectos sobre esta. Tiene por objetivo principal predecir la capacidad de respuesta necesaria que deberá tener la estructura, con la finalidad de resistir la gran variabilidad de acciones a las cuales se podrá ver afectada a lo largo de su vida útil.

Sobre las estructuras pueden actuar gran variedad de acciones de distinta naturaleza. Clásicamente, estas han sido las acciones gravitatorias (peso propio, sobrecargas de uso, cargas permanentes…), acciones térmicas como el flujo de calor, acciones reológicas como la retracción y la fluencia o acciones del terreno, como asientos o empujes. Estas acciones tienen como característica en común que varían lentamente a lo largo del tiempo, alcanzando su valor final en un periodo de tiempo suficientemente largo como para que esta variación no presente esfuerzos añadidos a la estructura, las llamadas fuerzas de inercia. No obstante, no todas las acciones son estáticas, sino que hay también las llamadas acciones dinámicas. Este tipo de acciones tienen como característica que su efecto varía a una gran velocidad, de modo que el efecto que provocan sobre la estructura es variable por cada instante de tiempo, lo que da lugar a la aparición de las fuerzas de inercia. Estas acciones son, por ejemplo, los sismos, el viento, el oleaje, las vibraciones, los impactos, las ondas de explosiones, etc. El estudio de estos efectos puede ser más importante que el de las acciones estáticas, lo que los convierte en determinantes a la hora de definir cuál debe ser la respuesta de la estructura.

Hasta la llegada del método de los elementos finitos (MEF), los problemas de estructuras se abordaban analíticamente, aplicando simplificaciones ingeniosas con la finalidad de poder resolverlos, condicionando los métodos de cálculo a problemas particulares. Por ejemplo, definiendo una geometría simplificada de la estructura o representando, mediante condiciones de contorno simples, sólo algunas de las acciones que las afectaban. Con su llegada, se ha permitido analizar estructuras

mucho más complejas y de una forma mucho más genérica, no sólo focalizando resolver un determinado tipo de problema. Esto afecta directamente a los problemas de acciones dinámicas, ya que mediante el método de los elementos finitos se ve facilitado enormemente su análisis.

En conclusión, en el análisis estructural es determinante estudiar cualquiera de las acciones o condiciones de contorno a las cuales puede estar sometida una estructura, independientemente de su naturaleza. La aparición del método de los elementos finitos supuso un gran avance a la hora de determinar su respuesta, ya que ha permitido mejorar la capacidad de análisis de los problemas.

1.2 MOTIVACIONES

Kratos [1] es una herramienta, desarrollada por CIMNE [2], que utiliza el método de los elementos finitos para poder llevar a cabo análisis numéricos y simulaciones de problemas de múltiples disciplinas de la ingeniería y de la ciencia, no sólo estructurales.

Actualmente, Kratos tiene la capacidad de aplicar multitud de condiciones de contorno en los distintos problemas por los que es empleado. En función de cuál sea su aplicación, se van a emplear un determinado tipo u otro. Por ejemplo, si el problema es de tipo estructural (como es el caso de la aplicación empleada en este trabajo), unas variables que se pueden implementar como condiciones de contorno son el desplazamiento o la fuerza, mientras que en otro problema, como puede ser uno de dinámica de fluidos, van a ser requeridas variables como la presión o la velocidad.

No obstante, esta herramienta no tiene la opción de aplicar las condiciones de contorno de forma que varíen en el tiempo, de forma que no se pueden representar acciones dinámicas. Hasta el momento sólo es posible implementar un determinado valor constante como condición de contorno, lo que vendría a representar las acciones estáticas.

Para tratar de analizar un problema con una condición de contorno variable, se tendría que realizar paso a paso la variación de la variable a lo largo del tiempo. Esto convierte el análisis de las acciones dinámicas en un proceso laborioso y costoso, y supone también que no se pueda determinar de forma más precisa la respuesta que debe tener la estructura.

1.3 OBJETIVOS

El objetivo principal de este trabajo es el de desarrollar una interfaz para Kratos con el lenguaje de programación Python [3]. Esta interfaz va a posibilitar que en Kratos se puedan introducir condiciones de contorno variables en el tiempo, de forma que se puedan analizar todo tipo de acciones dinámicas. Va a suponer una optimización del tiempo a la hora de analizar este tipo de problemas, además de proporcionar una respuesta mucho más precisa, sin importar en número de pasos o los datos a implementar.

Otro de los principales propósitos es que el diseño de la interfaz sea lo más genérico posible, posibilitando que se pueda aplicar el proceso a una gran variabilidad de casos y problemas ingenieriles, no sólo a problemas estructurales, de forma que podría implementarse en el proceso convencional de cálculo de Kratos. También se debe tratar que el diseño sea extensible, en el sentido en que si se cree conveniente, poder particularizar el proceso en determinados casos.

Se pretende comprobar su correcto funcionamiento, resolviendo ejemplos de problemas estructurales, contrastando los resultados obtenidos con resultados teóricos y con los de otros programas de elementos finitos.

Por otro lado, como objetivo adicional a la realización del trabajo, el autor va a tratar de aprender la metodología de cálculo de un problema mediante el método de los elementos finitos, a trabajar con Kratos como *problem type*, a utilizar el código de programación Python y el funcionamiento de GiD [4] como a pre y postprocesador.

1.4 ESTRUCTURA DE LA TESINA

Con el objetivo transmitir y exponer con claridad el trabajo realizado en la presente tesina, se ha creído que lo más conveniente era estructurarla de la siguiente forma:

1- Introducción. Se presenta el tema, las problemáticas que existen actualmente, se define la razón de ser de la tesina y se proponen los objetivos.

2- Estado del arte. Apartado en el que se realiza una revisión del estado de conocimiento actual sobre las herramientas a usar y la teoría que hay detrás.

3- Metodología de los elementos finitos. En él se describen las partes que conforman el método de los elementos finitos, claramente diferenciadas. También se describe como se realiza la imposición de las condiciones de contorno.

4- Proceso de imposición de condiciones de contorno variables. Sección en la que se describe el funcionamiento de la interfaz diseñada para Kratos, poniendo énfasis al porqué de su diseño y a los objetivos a cumplir.

5- Validación y ejemplos. En este apartado, se realizan una serie de ejemplos con el objetivo de comprobar el correcto funcionamiento de la interfaz creada. Los resultados obtenidos se comparan con las soluciones teóricas y con otros programas que usan también el método de los elementos finitos.

6- Conclusiones. Apartado donde se desarrollan las conclusiones del trabajo.

7- Agradecimientos. Se exponen los agradecimientos.

8- Referencias. Se citan las diversas referencias empleadas para la realización del presente documento.

CAPÍTULO 2

ESTADO DEL ARTE

La sección que se presenta a continuación tiene como misión realizar un repaso del estado de conocimiento actual sobre la temática a la cual pertenece este trabajo. Se va a definir la modelización de estructuras, poniendo énfasis en el método de los elementos finitos, ya que es el método que utiliza Kratos, la herramienta de cálculo que se va a usar. Seguidamente, se realiza un repaso de qué es y cómo se estructura. Por último, se describe qué tipo de modificaciones se pueden realizar en Kratos mediante procesos programados con el lenguaje de programación Python.

2.1 MODELIZACIÓN DE ESTRUCTURAS

El análisis del comportamiento mecánico de una estructura se lleva a cabo sobre modelos de ésta, entendiendo por modelo una idealización de la realidad física y funcional de la estructura de estudio.

Los modelos tienen como objetivo definir los esfuerzos, tensiones, movimientos y deformaciones a los cuales estará sometida la estructura. Para hacerlo, han de recoger la utilidad funcional del sólido, sus formas geométricas y su comportamiento.

2.1.1 El modelo físico

El primer paso en el proceso de análisis es el establecimiento de un modelo físico en el que se idealicen aquellas características físicas y funcionales de la estructura que participan en el comportamiento mecánico de esta.

Por tanto, ante la gran complejidad de la realidad física y funcional del sólido, se debe realizar una idealización parcial. Se deben analizar los factores más relevantes y que tendrán más influencia al realizar el cálculo, y se requerirá la utilización de hipótesis simplificadoras, con el objetivo de simplificar la complejidad del problema a estudiar. Una hipótesis simplificadora sería por ejemplo, la distribución continúa de la materia o la prevalencia del comportamiento macroscópico del sólido frente al comportamiento microscópico.

Tanto la selección de los aspectos a tener en cuenta en el modelo como las hipótesis simplificadoras que se pueden adoptar en el análisis son un proceso complejo, en el que va a ser determinante la experiencia del analista del problema.

2.1.2 El modelo matemático

A partir del modelo físico creado se desarrolla el modelo matemático, que consiste en un conjunto de variables y constantes relacionadas entre sí en un sistema de ecuaciones, todas ellas con unas ciertas condiciones iniciales y de contorno definidas. Las variables son las encargadas de modelizar el estado tensional y deformacional, la geometría y las características del modelo físico, mientras que las ecuaciones describen las relaciones entre estos.

Para definir el modelo se requiere la selección de un conjunto de grados de libertad. Los grados de libertad son componentes del movimiento del sólido que forman parte de la definición de su posición en el espacio en un instante y bajo un conjunto de acciones dadas.

Un sólido real es un continuo con infinitos grados de libertad, mientras que un modelo es un sistema discreto con un número finito de grados de libertad.

2.1.3 El modelo numérico

Un modelo numérico consiste, básicamente, en la implementación de un modelo matemático por medio de un ordenador. Se expresa, de forma común, como un algoritmo matemático.

Actualmente, debido al incremento de las capacidades y potencia de los ordenadores, se ha procedido a una mayor generalización de los algoritmos, sustituyendo los llamados métodos clásicos (que tenían técnicas especiales e ingeniosas para la resolución de problemas) por métodos basados en la teoría de matrices. Uno de estos métodos es el método de los elementos finitos.

Los métodos clásicos llevaban implícitas algunas simplificaciones que las hacían aplicables a estructuras con unas determinadas condiciones particulares. Con la aparición de los ordenadores y de los métodos basados en la teoría de matrices, se posibilitó el análisis de estructuras mucho más complejas, utilizando algoritmos de cálculo en los que no son necesarias tantas simplificaciones y aplicables a cualquier tipo de estructura.

2.2 INTRODUCCIÓN AL MÉTODO DE LOS ELEMENTOS FINITOS

El método de los elementos finitos (MEF) [6] es un método numérico que permite resolver diversos tipos de problemas relacionados con la ingeniería y con la ciencia. Tiene como finalidad obtener una solución numérica aplicada sobre un medio continuo.

La idea general que tiene el método es el de dividir un continuo en un conjunto de pequeños elementos interconectados por una serie de puntos, denominados nodos. Las ecuaciones que rigen el comportamiento del continuo, también van a cumplirse en los diferentes elementos creados. De esta forma se consigue pasar de un sistema continuo, con infinitos grados de libertad (GDL) y regido por una o un sistema de ecuaciones diferenciales, a un sistema con un número de grados de libertad finito, el comportamiento del cual se describe por un sistema de ecuaciones, lineales o no.

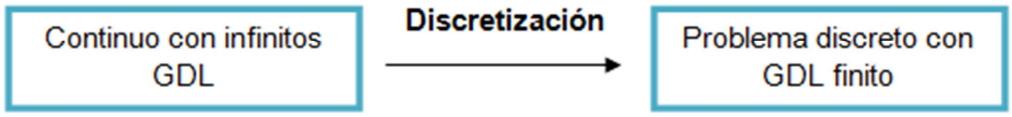

Figura 1. Esquema del proceso de discretización

En cualquier sistema, se puede realizar la siguiente clasificación geométrica:

- Dominio. Es el espacio geométrico donde se va a analizar el sistema.
- Contorno. Conjunto de puntos límites entre el dominio y el exterior.

En todo el espacio geométrico que representa el dominio se deben cumplir las ecuaciones de gobierno. Por su parte, las condiciones de contorno son ecuaciones las cuales definen el valor de la variable objeto de estudio o de su derivada en el contorno del dominio.

Figura 2. Idealización de la clasificación geométrica de un sistema

Se supone que el dominio es discretizado a través de una malla en subdominios, denominados elementos. La división puede ser mediante puntos, líneas o superficies, en función de la dimensión del caso de estudio. Los elementos se definen por un número discreto de puntos, los nodos, los cuales los conectan entre ellos. Es sobre estos nodos donde se materializan las incógnitas fundamentales del problema. En el caso de un elemento estructural, una de las incógnitas podría ser el desplazamiento de los nodos. Estas incógnitas, los denominados grados de libertad, determinan el estado y/o la posición del nodo.

En conclusión, el MEF es un método que satisface unas necesidades de cálculo, en general muy complejas, que con los métodos de cálculo tradicionales eran muy difíciles de cumplir. Permite realizar un cálculo de una forma más sistemática, y a la práctica, es un sistema adecuado de análisis que determina las bases idóneas para el desarrollo de *software*.

2.3 KRATOS

2.3.1 ¿Qué es Kratos?

Kratos [1] es una herramienta, desarrollada en CIMNE [2] y con un código abierto en lenguaje C++, que se usa para llevar a cabo simulaciones y análisis numéricos multidisciplinares de elementos finitos. Proporciona varios instrumentos para una sencilla puesta en práctica de las aplicaciones de elementos finitos, además de una plataforma común que proporciona una interacción fácil entre ellas.

Puede ser definido por cinco características que lo identifican. Kratos es:

- Multifísico. Engloba en un solo *software* la capacidad para realizar análisis de varios problemas de ingeniería, como son los estructurales, los de dinámica de fluidos y los termodinámicos.
- Un método de elementos finitos (MEF). Gracias a los métodos numéricos, soluciona muchos problemas que surgen en problemas ingenieriles y científicos. El método de elementos finitos es uno de los más poderosos, flexibles y versátiles métodos que existen.
- Orientado a objetos. Es de carácter general, flexible y reusable, de forma que permite afrontar gran variedad de retos en métodos numéricos.

- Abierto. El código y la estructura del programa está disponible por si cualquier usuario está dispuesto a ampliarlo.
- Libre. Está dedicado principalmente para desarrolladores, investigadores y estudiantes, lo que lo convierte en un modo muy fructuoso de compartir el conocimiento.

2.3.2 Estructura de kratos

Como se ha comentado en el apartado anterior, Kratos no está diseñado como un código monolítico, sino que se puede aplicar a varios problemas muy diferentes entre sí. Viene a ser como una biblioteca donde los usuarios pueden encontrar y combinar las diferentes herramientas necesarias para resolver un problema en particular.

2.3.2.1 Kernel y aplicaciones

Kratos puede ser dividido en dos categorías: Kernel y aplicaciones.

El Kernel consiste en el núcleo, el centro del cálculo. Proporciona la infraestructura básica y las herramientas numéricas generales, es decir, constituye de base sobre el cual se construyen las diferentes aplicaciones.

Las aplicaciones consisten en una adaptación de Kratos respecto de la temática del problema a resolver. Una aplicación proporciona una implementación de un conjunto de algoritmos utilizados en la simulación de problemas en un campo determinado, como la dinámica de fluidos o la mecánica de sólidos. En caso de problemas de temáticas muy variadas, el Kernel también permite que haya una cierta interacción entre diferentes aplicaciones.

La principal ventaja que proporciona esta estructura es que hay una clara separación entre la base numérica del código y las partes que están enfocadas a la simulación de un problema en particular. De esta forma se evitan conflictos en el desarrollo de distintas aplicaciones, ya que los desarrolladores de Kratos pueden concentrarse en la ampliación del código sin preocuparse de cometer errores en otros campos, además de recudir el tiempo de compilación.

Debido a que en este trabajo se van a analizar problemas estructurales, la aplicación de Kratos que se va a usar es *Solid Mechanic Application;* pero hay otras, como *Incompressible Fluid Application*, para problemas de dinámica de fluidos y *Convection Diffusion Application*, para problemas relacionados con la variación de temperatura (convección y dilatación).

Todas las herramientas usadas por Kratos están compiladas en C++, pero tienen una interfaz para ser llamadas con Phython [3], lo que permite a los usuarios realizar modificaciones. Aquí es donde aparece el concepto de procesos y de *utilities*, que son clases que se pueden utilizar para implementar estas modificaciones comentadas en la simulación.

Un proceso es una herramienta que se utiliza para implementar tareas generales, pero que no están cubiertas en el proceso convencional de iteración de la solución.

Las *utilities* son un conjunto de herramientas que se utilizan para llevar a cabo una tarea en particular, o para algunas tareas con temas en común, como por ejemplo funciones matemáticas o herramientas de paralelización.

2.3.2.2 Funcionamiento de la aplicación *Solid Mechanic*

Solid Mechanic Application es una aplicación de Kratos utilizada para resolver problemas estructurales. En función del tipo de estructura a resolver, el usuario puede particularizar el problema entre distintos tipos de elementos que están disponibles.

Se puede estructurar de la siguiente forma:

- Tipo de análisis

Tipo de análisis que se quiere llevar a cabo en función del tipo de estructura a estudiar. Los distintos tipos a analizar son:

  - *Beam*. Para estructuras tipo viga. Para análisis en 3 dimensiones.
  - *Shell*. Para estructuras tipo placa, con dos direcciones principales. Resisten tensiones de membrana, esfuerzos de flexión y esfuerzos de cortante.
  - *Solid*. Para estructuras discretizadas internamente. El cuerpo tiene que estar mallado con elementos (triángulos o cuadriláteros en análisis 2D y tetraedros y hexaedros en 3D).
  - Membrana. Para estructuras mixtas, compuestas por partes de los otros tipos descritos anteriormente.

- Tipo de cinemática:

  - Pequeños desplazamientos.
  - Grandes desplazamientos.

- Opciones de tipo cinemático:

  - Estático. El equilibrio de la estructura se halla a partir de las condiciones de contorno, las cargas y los materiales escogidos. La deformación es única.
  - Dinámico. El equilibrio se encuentra mientras se incrementan las cargas o el tiempo.
  - Cuasi estático. El equilibrio se encuentra mientras se incrementan las cargas o el tiempo, pero la inercia no cambia.

- Opciones del tipo de análisis:

  - Lineal. El equilibrio de la estructura se calcula resolviendo primero un sistema de ecuaciones primero, despreciando todas las posibles consecuencias de la geometría, cargas, condiciones de contorno y el estado del material.
  - No lineal. El equilibrio de la estructura se calcula iterativamente, actualizando la geometría, cargas, condiciones de contorno y el estado del material, hasta que se lleva a converger.

- Condiciones de contorno

Se pueden introducir dos tipos diferentes de condiciones de contorno:

  - Desplazamientos. Un desplazamiento inicial impuesto puede ser fijado en cualquiera de las tres direcciones (X, Y, Z), y puede ser aplicado a puntos, líneas, áreas o volúmenes.
  - Rotaciones. Una rotación inicial impuesta puede ser fijada en cualquiera de las tres direcciones (X, Y, Z), y puede ser aplicada a puntos, líneas o áreas. Sólo se puede aplicar a elementos tipos *Beam* y tipo *Shell* sólo si el elemento es isotrópico.

- Cargas

Las cargas que se pueden aplicar son:

  - Peso propio. El efecto de la gravedad puede ser definido en cualquier dirección del espacio, y se puede aplicar a puntos, líneas, superficies o volúmenes.

- Carga. Las cargas pueden ser puntuales (aplicadas a puntos) o repartidas (aplicadas a líneas o superficies).
- Momento flector. Sólo se pueden aplicar a puntos.
- Carga de presión. Es un tipo de carga que siempre actúa en dirección perpendicular a la superficie. Puede ser aplicada en líneas o superficies.

- Leyes constitutivas

Por ahora, se pueden realizar dos modelos. El primero, es un modelo elástico, y el segundo, más complejo, es un modelo elasto-plástico. Este último usa el criterio de Von Mises.

2.4 PYTHON

Python es un lenguaje de programación caracterizado por tener una sintaxis muy limpia y simple. Es utilizado por Kratos con el objetivo de combinar de una forma más flexible y dinámica sus componentes al realizar diferentes tipos de simulaciones. El *script* principal de Python interpreta el papel de la función principal del programa, pero con una gran facilidad de acceso tanto de usuarios como desarrolladores.

La gran ventaja de estos *scripts* sobre el código compilado es que los cambios se pueden hacer muy rápidamente. Por ejemplo, se puede cambiar la tolerancia de un solucionador o también cambiar unas condiciones de contorno. Por otro lado, hay que tener en cuenta que el proceso será más lento, así que se ha de estudiar su utilización.

En este sentido, los *scripts* de Python pueden realizar las siguientes acciones:

- Cargar Kratos e importar la aplicación.
- Leer los datos del problema y crear y modificar el *model part* de Kratos, que son la malla, elementos, materiales y las condiciones de contorno.
- Definir el constructor y solucionador.
- Llamar al solucionador.
- Escribir un archivo con la malla y los resultados.

Por tanto, una vez compilado Kratos, ya se ha usado todo el código programado en C++ necesario para la aplicación. El resto de tareas pueden ser realizadas por Python,

con las ventajas ya mencionadas. El *script* que se crea actúa como interfaz o proceso entre el guión del problema y las funciones de Kratos, añadiendo funciones adicionales a realizar en el cálculo que no se realicen de forma convencional.

CAPÍTULO 3

METODOLOGÍA DEL MÉTODO DE LOS ELEMENTOS FINITOS

En el presente capítulo se va a presentar cual es el método de trabajo del método de los elementos finitos, particularizado para el caso de Kratos. El objetivo es que quede de forma clara y detallada como se debe proceder para llevar a cabo la resolución de un problema. En la segunda parte del capítulo, se presenta como se realiza la imposición de las condiciones de contorno en un problema de este tipo.

3.1 MÉTODO DE TRABAJO DEL MÉTODO DE LOS ELEMENTOS FINITOS

Al utilizar para la resolución de un problema el método de elementos finitos, se puede dividir el proceso en tres etapas claramente diferenciadas.

- Preproceso
- Cálculo
- Postproceso

El esquema de trabajo a utilizar, con la herramienta Kratos [1], es el siguiente:

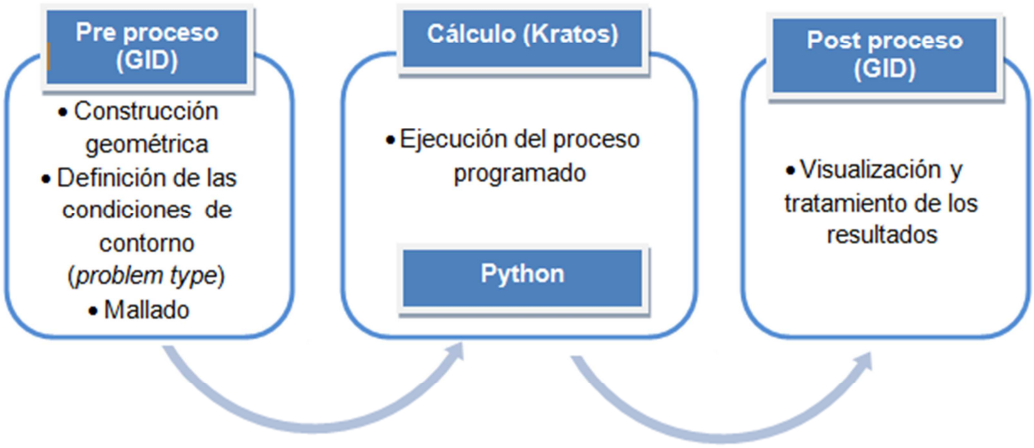

Figura 3. Esquema de trabajo del método de los elementos finitos

A continuación, en los siguientes apartados, se procede a realizar una explicación de cuáles son las características de cada etapa del proceso.

3.1.1 Preproceso

Es la etapa inicial, en la que se definen la geometría, las condiciones de contorno, las propiedades de los materiales y se genera la malla.

Las labores del preproceso son abordadas mediante GiD [4], una poderosa herramienta de CAD desarrollada por CIMNE [2] que permite una eficiente gestión integral de las labores del preprocesado, tanto en modelos bidimensionales como tridimensionales.

GiD genera una serie de archivos con todas las condiciones comentadas y con los datos de la malla, los cuales van a ser usados en la fase de cálculo.

3.1.1.1 Definición de la geometría

A partir de una idealización de la estructura a estudiar, se dibujan puntos, líneas, áreas y volúmenes. Se sigue una estructura jerárquica para su definición, siendo necesarios los puntos para definir las líneas, las líneas para definir las superficies y las superficies para poder generar los volúmenes.

En consecuencia, un elemento en dos dimensiones va a ser definido mediante puntos, líneas y superficies, mientras que un elemento en tres dimensiones va a necesitar, además, definir un volumen.

3.1.1.2 Definición de las condiciones de contorno y de los materiales

En función del cálculo a realizar, se van a introducir unas determinadas condiciones de contorno u otras. Es una parte trascendental antes del cálculo a llevar a cabo.

GiD permite adaptar el tipo del problema al cálculo a realizar, mediante la creación de un *problem type,* que en éste caso es Kratos. Activando su interfaz aplicada a estructuras, *Solid Mechanic Application*, se definen todas las características que se necesitan para el cálculo; se particulariza el problema definiendo las características de los materiales, introduciendo las cargas a las cuales está sometido el elemento de estudio y se definen sus condiciones de contorno.

3.1.1.3 Mallado

Para la aproximación de la solución de ecuaciones diferenciales, el método de los elementos finitos requiere que se realice una discretización del modelo. Por tanto, previamente a proceder a la etapa de cálculo, se debe realizar el mallado, dividiendo la estructura en partes más pequeñas.

Se generan una serie de puntos, los nodos (explicados en el capítulo anterior), que constituyen la forma del diseño. Están interconectados entre sí, y contienen las características del material y todas las propiedades estructurales del modelo. El conjunto de elementos de discretización es lo que se conoce como malla. Básicamente se pueden clasificar las mallas en dos tipos: las estructuradas y las no estructuradas.

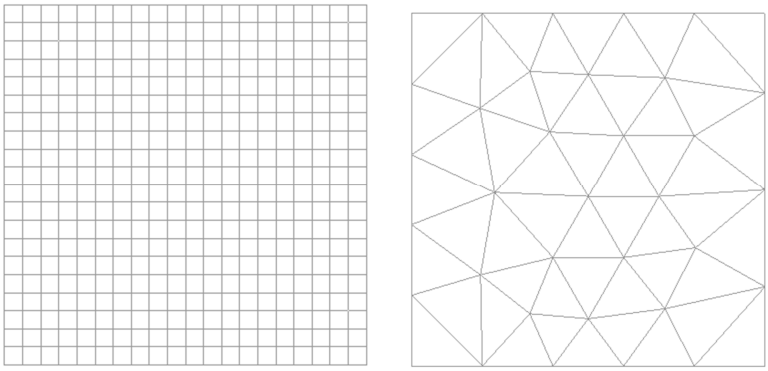

Figura 4. Ejemplo de malla estructurada y no estructurada

La calidad de la malla es un aspecto muy relevante en la bondad de los resultados numéricos obtenidos, debiendo analizarse detenidamente. Es para esta razón que la densidad de la malla puede variar a lo largo del área o volumen del elemento a estudiar. Las zonas que sean singulares y tengan puntos importantes de interés van a tener un mallado más fino en comparación con el resto del modelo.

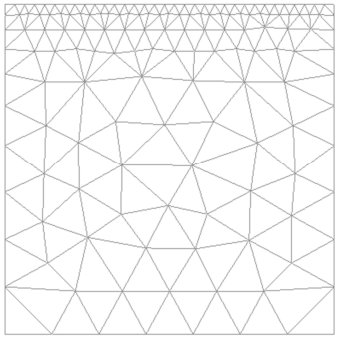

Figura 5. Ejemplo de mallado más fino en una zona determinada

Es importante tener en cuenta que, cuanto más densa sea la malla, más se prolonga el coste computacional, así que es importante definir y estudiar bien previamente cuales van a ser éstas zonas singulares de estudio, con el fin de optimizar el rendimiento, reduciendo el tiempo de cálculo obteniendo unos buenos resultados.

3.1.2 Cálculo

Etapa en la que el programa de cálculo resuelve el problema descrito en el preproceso y que genera las soluciones. En función del cálculo a realizar, se debe utilizar un *problem type*, que en este caso es Kratos.

Es determinante el tipo de cálculo escogido a realizar (por ejemplo, si es un problema estático o dinámico) y una buena configuración de los parámetros de cálculo, como por ejemplo la selección de los intervalos de tiempo, el número de iteraciones o la norma del error.

Una vez se inicia el cálculo, Kratos empieza transfiriendo las cargas al modelo, genera las matrices de rigidez, realiza la triangulación de la matriz, resuelve el sistema de ecuaciones y genera la solución.

Al ejecutarlo en GiD, se generan los siguientes ficheros fruto de este cálculo realizado:

- <nombre fichero>.cnd
- <nombre fichero>.geo
- <nombre fichero>.info
- <nombre fichero>.msh
- <nombre fichero>.prb
- <nombre fichero>.rdr
- <nombre fichero>.lin
- <nombre fichero>.mdpa
- <nombre fichero>_var.py
- out.out
- script.py

Estos archivos son los que emplea Kratos para resolver el problema. El archivo *script.py* es un archivo escrito en Python que permite ejecutar procesos o *utilities* programados no cubiertos en el proceso convencional de iteración de la solución, como ya se ha comentado en los apartados 2.3 y 2.4.

3.1.3 Postproceso

Es la etapa final donde se pueden visualizar los resultados obtenidos en el cálculo, representados con gráficas para una mejor comprensión. Debe incluir una fase de comprobación con el fin de verificar los resultados.

En este caso, la herramienta usada vuelve a ser GiD. En el manual de referencia se pueden observar la gran variedad de presentación de los resultados que se pueden obtener [5].

3.2 APLICACIÓN DE LAS CONDICIONES DE CONTORNO

Las condiciones de contorno son las que definen el modelo en sus límites, y son necesarias para la resolución numérica del sistema discretizado mediante derivadas parciales.

En la aplicación del método de elementos finitos, existen dos tipos de condiciones de contorno, las de tipo Dirichlet y las de tipo Neumann.

Una condición de contorno es tipo Dirichlet si, siendo $u(x, y, z, t)$ la función a determinar, la condición impuesta en una determinada zona situada en el contorno $u(x_o, y_o, z_o)$, es del tipo $u(x_o, y_o, z_o, t) = \phi(t)$. Este tipo de condición se aplica a los nodos y se fija directamente a los grados de libertad del elemento. Por ejemplo, sería una condición de este tipo un desplazamiento impuesto.

Una condición de contorno es de tipo Neumann si, siendo $u(x, y, z, t)$ la función a determinar, la condición impuesta en una determinada zona situada en el contorno $u(x_o, y_o, z_o)$, es del tipo $\frac{\partial u}{\partial x}(x_o, y_o, z_o, t) = \Psi(t)$, $\frac{\partial u}{\partial y}(x_o, y_o, z_o, t) = \Psi(t)$ o $\frac{\partial u}{\partial z}(x_o, y_o, z_o, t) = \Psi(t)$. Este tipo de condición también se aplica en los nodos, pero no se fija directamente a los grados de libertad del elemento, sino que genera un objeto condición para que la aplique. Por ejemplo, sería una condición de este tipo una carga puntual.

En el dominio de análisis Ω, se define Γ como el contorno de Ω. Considerando Γ_u como el contorno de las condiciones de contorno de tipo Dirichlet, y Γ_q como el contorno de las condiciones de tipo Neumann, debe cumplirse que:

$$\Gamma_u \cup \Gamma_q = \Gamma$$

$$\Gamma_u \cap \Gamma_q = 0$$

En el proceso creado para Kratos que se presentará en el capítulo de a continuación, la variable de una condición de contorno es independiente que sea de tipo Dirichlet o que sea de tipo Neumann. Se podrán aplicar, indistintamente, desplazamientos, fuerzas aplicadas, rotaciones impuestas… Este proceso ofrece la posibilidad de aplicar condiciones de contorno variables en el tiempo. Las condiciones de contorno pueden ser fruto de la acción, por ejemplo, de un sismo, viento, impacto, oleaje, vibración, etc.

CAPÍTULO 4

PROCESO DE IMPOSICIÓN DE CONDICIONES DE CONTORNO VARIABLES

En este capítulo se van a precisar cuáles son los objetivos y el diseño de la interfaz creada; y por último, el desarrollo de su funcionamiento y de cómo debe implementarse en la metodología de cálculo de Kratos [1].

4.1 OBJETIVOS

El objetivo principal que se ha querido reflejar es el de crear un proceso que pueda aplicar condiciones de contorno variables en el tiempo, como ya se ha introducido en el final del capítulo anterior. Las características básicas que debe tener este proceso es que sea genérico y muy extensible.

Genérico porque se pretende que sea aplicable a cualquier tipo de problema físico, no focalizarlo en un solo tipo de problema.

Extensible porque se pretende que sea un proceso modificable por otros usuarios, añadiendo las condiciones y variaciones que se crean convenientes.

Por tanto, la idea general es que este proceso sea tan flexible como Kratos: multifísico, adaptable a múltiples campos y permitiendo modificaciones cuando sean necesarias, particularizando el proceso.

4.2 DISEÑO

Con la finalidad de cumplir los objetivos mencionados, se ha diseñado la interfaz de forma que:

- A partir del *problem type* correspondiente, que en este caso es Kratos, se obtengan los datos necesarios para poder utilizar el proceso.
- Se puedan aplicar las condiciones de contorno variables en el tiempo independientemente de cuál sea la variable o la naturaleza de estas. Se pueden imponer desplazamientos, velocidades, aceleraciones, rotaciones, cargas puntuales, etc.
- El formato de entrada es en forma de tabla tiempo – valor de la variable, ya que es una forma muy sencilla de implementar las condiciones sobre los nodos correspondientes. En el caso que la opción de entrada sea una función respecto del tiempo o un valor constante simplemente, el proceso tiene mecanismos para transformarlo en el formato requerido.

4.3 IMPLEMENTACIÓN DE LA INTERFAZ

En este apartado se define el desarrollo que se lleva a cabo en la interfaz. La interfaz creada básicamente es el proceso llamado *apply_nodal_variable_process,* aunque también se ha desarrollado una particularización de este, el proceso *ground_motion_process*, apto para problemas de tipo sísmico. A continuación, se va a determinar cuáles son los pasos y las modificaciones a seguir con la finalidad de introducir la interfaz en el proceso de cálculo de Kratos.

4.3.1 Aplicación de condiciones de contorno variables a nodos

El proceso a usar es llamado *apply_nodal_variable_process*. La función que realiza es la de imponer, a cada paso de tiempo, un determinado valor a los nodos previamente seleccionados. Por tanto, para cada instante de tiempo el valor de la variable va a ser actualizado.

A continuación, se describe el procedimiento que debe realizar el usuario para utilizarla.

Primero de todo, se deben introducir los parámetros obligatorios, lo que significa que no tienen valores por defecto. Hay que definir cuál es la variable de estudio, el tiempo de inicio y el tiempo final de la aplicación de la condición de contorno, y los nodos a los que se les va a imponer la condición de contorno.

A continuación, ya se pueden introducir los parámetros opcionales. Para estudiar cuál es la respuesta de la estructura, el usuario tiene tres distintas opciones, como ya se ha introducido en el apartado anterior. Son las siguientes:

- Determinar un valor constante a lo largo del tiempo.
- Insertar una tabla tiempo – valor.
- Definir una función que defina la variación de la variable de estudio respecto del tiempo.

También se puede definir, adicionalmente, una función respecto del espacio.

Una vez introducidos todos los datos de entrada necesarios, el procedimiento que sigue a cabo la interfaz es el siguiente:

1- Se define automáticamente un tiempo actual, que corresponde al instante de tiempo en el que se encuentra la simulación.
2- La aplicación de la condición de contorno variable empieza cuando éste tiempo coincide con el tiempo de inicio previamente introducido. Para instantes de tiempo anteriores no se aplicará ninguna.
3- Al empezar el proceso busca cual de las tres opciones, definidas anteriormente, es la introducida.
   Los datos más sencillos de utilizar por el fichero son los de tabla tiempo – valor y los de valor constante a lo largo del tiempo, ya que éste trabaja con filas y columnas de series de datos. En cambio, si el dato introducido es una función variable en el tiempo, se debe llamar a un subproceso, llamado *get_factor*, el cual transforma dicha función en una tabla tiempo – valor, apto para ser usado por el proceso. El intervalo de tiempo entre cada valor de la tabla creada es definido por el usuario.
4- En el caso de que se haya definido, adicionalmente, una función respecto del espacio, el valor de entrada de la tabla se ve modificado por ésta.
5- Con la serie de datos cada un determinado tiempo, se fija el valor obtenido para cada respectivo nodo de la estructura, finalizando cuando se llega al tiempo final de aplicación de la condición de contorno.
6- Llegado a este instante de tiempo, el usuario puede escoger si quiere obtener algún valor o no hasta el tiempo final de simulación, ya que no tienen porqué coincidir con el tiempo final de aplicación de la condición de contorno. Esto se realiza mediante el parámetro denominado *keep_after_end*. Si este valor se le define como *False*, a cada nodo de estudio se le aplicará el valor por defecto,

que inicialmente se haya autorizado, hasta este tiempo final de simulación. Éste valor por defecto se le ha asignado un valor de 0.0, si es que el usuario no decide cambiarlo. En caso contrario, si se define como *True*, no se asignará ningún valor por defecto, sino que se mantendrá el último valor hasta el final de la simulación.

En el apéndice 1 se puede encontrar la programación de este proceso.

4.3.2 Aplicación de condiciones de contorno variables sísmicas a nodos

En el caso en el que la condición de contorno variable a imponer sea una variable sísmica, se ha hecho una particularización del proceso *apply_nodal_variable_process*.

La problemática se encuentra en que la variable de entrada pueden ser la velocidad o la aceleración, no sólo el desplazamiento, como se explica en el libro “Estructuras sometidas a acciones sísmicas” [7]. Por tanto, aunque el usuario lo que quiera sea aplicar desplazamientos en unos nodos determinados para así estudiar cual es el desplazamiento global de la estructura, puede ser que no tenga directamente los datos en una tabla tiempo - desplazamiento, sino que los tenga con la velocidad o la aceleración como variables. En el caso de que el dato de entrada sea la velocidad, se debe realizar una integración para así obtener el desplazamiento correspondiente; y en el caso de que el dato de entrada sea la aceleración, se debe realizar una doble integración.

Es por este motivo que se ha particularizado el proceso *apply_nodal_variable_process*, para que pueda aceptar como datos de entrada la velocidad o la aceleración. Este nuevo proceso es llamado *ground_motion_process.*

El proceso *ground_motion_process* tiene únicamente como opción de formato de entrada una tabla. Además, se debe especificar sobre qué tipo variable es el dato se está introduciendo (desplazamiento, velocidad o aceleración) y sobre cuál es su componente (X, Y o Z). El resto de variables a introducir son los mismos.

Es importante remarcar que los datos de entrada tienen que estar en formato tabla. En el caso de que estos datos estén en formato lista, como los de OpenSees, se ha creado una función, llamada *convert_peer_motion_to_table*, que estructura estos datos en forma de tabla (se explica más detalladamente en el apartado siguiente de este capítulo).

Una vez introducidos todos los datos de entrada necesarios, lo primero que se analiza es cuál es la variable introducida. Como se ha comentado ya anteriormente, en el caso de que las variables de entrada sean la velocidad o la aceleración, la integración se deberá realizar una o dos veces, respectivamente. En el caso de que la variable sea el desplazamiento, la integración no es necesaria.

La integración se aproxima mediante el método del trapecio. Éste método consiste en aproximar cada subintervalo de la integral por el área de un trapecio definido por los puntos de lo delimitan. El subintervalo consiste en cada instante de tiempo analizado. Se debe destacar que esta no es una buena aproximación y que los resultados de desplazamiento obtenidos no van a ser los esperados, hecho que ya se explica en la web de OpenSees [8]. La aproximación se ha realizado para demostrar que es posible aplicar la aceleración o la velocidad como variables, no es necesario que sea el desplazamiento la variable a introducir. En el caso de que el usuario quiera realizar un análisis con resultados próximos a la realidad, se deberá introducir manualmente otro método que realice la integración que sustituya al método del trapecio.

En conclusión, el usuario introduce una tabla Tiempo - Desplazamiento/Velocidad/Aceleración, y el proceso *ground_motion_process* la transforma en una tabla Tiempo – Desplazamiento apta para poder utilizar el proceso *apply_nodal_variable_process* y poder aplicar ésta condición de contorno variable a los nodos de la estructura de estudio.

En el apéndice 2 se puede encontrar la programación de este proceso.

4.3.2.1 Función peer_motion_to_table

Como ya se ha mencionado, el formato de entrada para los procesos creados es un valor constante en el tiempo, una función o una tabla tipo tiempo – valor. Cuando la opción de entrada no sea una de estas tres opciones, la clase no podrá leer los datos de entrada.

Esta función es creada para resolver una problemática de este tipo. Su objetivo es el de transformar series de datos que estén en tipo lista, colocados de forma consecutiva cada un cierto intervalo de tiempo, en una tabla tiempo – valor, apta para ser introducida como formato de entrada.

En este caso se ha particularizado para el ejemplo de OpenSees, pero modificándose convenientemente se pueden transformar todo tipo de listas en tablas tiempo – valor.

En el apéndice 2.1 se puede encontrar la programación de esta función.

4.3.3 Implementación de la interfaz en el cálculo

Los procesos creados son archivos escritos en Python [3] que van a tener que ser introducidos en la carpeta <nombre.fichero>.gid creada en el preproceso.

Al no formar parte del cálculo convencional que hace Kratos en el proceso de iteración para obtener la solución, cuando desde GiD [4] se ejecute el cálculo éste no usará los ficheros creados. Para que sí se usen los ficheros en el cálculo, primero se deben hacer unas modificaciones en los archivo creados *KratosStructuralMP* y *ProjectParameters*.

En el archivo *ProjectParameters*, se debe actualizar el *solver_type* con el "*mechanical_solver_V2*".

En el archivo *KratosStructuralOpenMP,* las modificaciones a llevar a cabo son:

1- Cambiar el archivo *conditions_python_utility* por *conditions_python_utility_mod.*

2- Crear el proceso personalizado (*custom process*), justo antes de que se inicialice el *solver*.
Este proceso personalizado utiliza los procesos creados para poder solucionar el problema. En él se definen las características requeridas para poder ejecutarlo correctamente. Por ejemplo, se debe definir cuál es la variable de estudio, cuál es el formato de entrada de los datos o cuáles son los nodos a aplicar las condiciones de contorno.

3- En el bucle, se define cuando inicializar y finalizar el proceso propio, por cada paso de tiempo. El texto que se introduce es el siguiente:

```
#######################################################################
  # 2 INITIALIZE CUSTOM PROCESSES
  for iProc in customProcesses:
      iProc.ExecuteInitializeSolutionStep()
#######################################################################

#######################################################################
  # 3 FINALIZE CUSTOM PROCESSES
  for iProc in customProcesses:
        iProc.ExecuteFinalizeSolutionStep()
#######################################################################
```

La modificación 2 del archivo *KratosStructuralOpenMP* es la única que variará en función del problema a resolver. Todas las otras van a ser las mismas, independientemente del problema

Una vez realizadas todas las modificaciones necesarias, se debe ejecutar por comandos el *runkratos.exe*. Este es un archivo que se encuentra en la carpeta de Kratos como *problem type,* y es el responsable de ejecutar el cálculo de los problemas. Por tanto, ejecutándolo des de la carpeta donde se encuentra el problema se puede recalcular, con la novedad de las condiciones introducidas en el proceso.

Después del tiempo de cálculo, accediendo al postproceso desde GiD, van a poder observarse los resultados obtenidos.

En el apéndice 3 se puede encontrar archivo *KratosStructuralOpenMP* particularizado para la resolución del ejemplo 5.5.1.

CAPÍTULO 5

VALIDACIÓN Y EJEMPLOS

Con el objetivo de validar el correcto funcionamiento de la interfaz creada para Kratos [1], en este capítulo se van a calcular una serie de ejemplos seleccionados. Se va a tratar de utilizar distintos formatos de entrada, funciones o bien tablas; y diferentes variables, como por ejemplo una carga puntual o un desplazamiento.

Primeramente, se van a calcular dos ejemplos básicos con el objetivo de visualizar si la condición de contorno variable es aplicada correctamente.

A continuación, una vez validada la correcta imposición de las condiciones, se van a analizar algunos problemas de más complejidad. Los resultados obtenidos se van a comparar, según el caso, con soluciones teóricas o con los programas XFINAS y OpenSees, los cuales también usan el método de los elementos finitos.

5.1 PILAR SOMETIDO A UNA CARGA PUNTUAL

5.1.1 Descripción del problema

La estructura de análisis consiste en un pilar de 5m de longitud, empotrado en un extremo y libre en el otro. La sección es cuadrada, de 0,3x0,3m, y las características mecánicas del material son las siguientes:

Módulo de Elasticidad	$2{,}0\cdot10^{11}$	N/m^2
Número de Poisson	0,29	-
Densidad	7860	Kg/m^3

Tabla 1. Características mecánicas

La carga a la que va a estar sometida la estructura es a una carga puntual P variable en el tiempo, de valor:

$$P(t) = -10\mathrm{t}$$

donde t es el tiempo.

El efecto de esta carga se va a analizar para un tiempo total de 50s.

Con la finalidad de ver directamente cual es el efecto de esta carga sobre la estructura, se va a despreciar el efecto del peso propio.

El esquema estructural de la estructura a resolver es el siguiente:

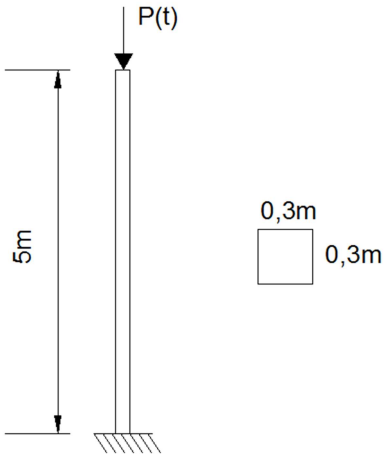

Figura 6. Esquema estructural

El objetivo de este ejemplo es el de comprobar que la carga se esté aplicando correctamente en el nodo correspondiente durante los 50 segundos de simulación.

5.1.2 Procedimiento a seguir para la resolución del problema

Se introduce la geometría dada en GiD [4].

Figura 7. Geometría

Declarando Kratos como *problemtype*, se puede empezar a aplicar las condiciones a la estructura. Se define el material de estudio, con las propiedades mecánicas anteriormente descritas.

A continuación, se deben definir las características del modelo.

- Tipo de análisis

  El tipo de análisis a realizar es genérico, dinámico, lineal y enfocado a pequeños desplazamientos.

- Estrategia de solución

  Se define como tiempo total 50s, con un paso de 0,1s.

- Propiedades

  En este apartado se escoge el material anteriormente creado, la sección cuadrada de 0,3x0,3m y el tipo de propiedad, que en este caso es *Beam,* ya que se trabaja con un pilar. Con todas estas características se define una propiedad.

  La propiedad creada es la que se asigna al elemento dibujado, el pilar.

- Condiciones de contorno

  Al estar la estructura empotrada en su base, se va a imponer en el nodo que tanto el desplazamiento como la rotación sean nulas en cualquier de los tres ejes X, Y o Z.

- Cargas

  La carga puntual P, al ser función del tiempo, no es una condición que Kratos pueda generar desde su aplicación *Solid Mechanical*. Se utilizará la interfaz creada en Python para poder introducir dicha carga. Para poder implementarla, primero se debe aplicar la carga en el instante inicial en el nodo de aplicación, de modo que Kratos genere que en este va a haber una carga aplicada; luego se tendrá que realizar un “precálculo”, de forma que se creen los archivos Python [3] en la carpeta del problema, sobre las cuales se van a realizar las correspondientes modificaciones.

Antes del “precálculo”, pero, se debe generar el mallado del elemento. En este caso se trata de una malla no estructurada triangular, compuesta de 11 nodos y de 10 elementos.

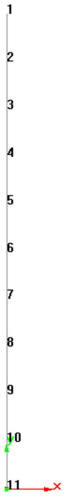

Figura 8. Mallado

Ahora sí que se puede realizar el “precálculo”. En la carpeta donde se almacenan los ficheros del problema de estudio veremos que se obtienen los archivos *KratosStructuralOpenMP y ProjectParameters*. Es sobre estos archivos, como se ha explicado en el capítulo 4, donde se podrán realizar las modificaciones convenientes para así poder introducir las cargas variables en el problema.

En este caso, el proceso personalizado (*custom process*) que se crea en el archivo *KratosStructuralOpenMP* es el siguiente:

```
##################################################################
# 1 CREATE CUSTOM PROCESSES
import apply_nodal_variable_process

def user_time_function_proc(t):
        return -10*t

customProcesses = []

class nodal_variable_settings:
    # these 2 parameters are for the generation of the process
    file_name = "apply_nodal_variable_process"
    class_name = "apply_nodal_variable"
    group_id = 0
    # parameters used by this custom process
    fix = True
    user_time_function = user_time_function_proc
    value = None
    user_spatial_function=None
    time_table = None
    time_begin = 0.0
```

```
    time_end = 50.0
    default_value = 0.0
    keep_after_end = False
    nodes = [1]
    variable_name = "POINT_LOAD_Y"

proc1 = apply_nodal_variable_process.CreateCustomProcess(model_part,
nodal_variable_settings)
customProcesses.append(proc1)
#######################################################################
```

Como se puede ver:

- Se crea la clase *nodal_variable_settings*.
- Se utiliza el proceso *apply_nodal_variable_process* para poder aplicar la carga correctamente.
- La variable de estudio es llamada “POINT_LOAD_Y”, ya que la carga es puntual y en esta dirección.
- Se define el nodo de aplicación, el número 1, punto correspondiente al extremo superior libre del pilar.
- Se define la función P(t), llamada *user_time_function_proc,* como dato de entrada.
- Se establecen el tiempo de inicio y final de aplicación de la carga.

Con esta modificación, junto con las otras generales, ya se puede ejecutar el runkratos.exe y realizar el cálculo del problema.

Una vez realizado, accediendo al apartado de postproceso de GiD, ya se pueden ver los resultados obtenidos.

5.1.3 Análisis de los resultados obtenidos

Inicialmente, se comprueba que las carga aplicada han sido definida correctamente en la posición esperada.

Figura 9. Punto de aplicación y dirección de la carga puntual

La evolución de la carga aplicada a lo largo del tiempo debe cumplir la función $P(t) = -10\mathrm{t}$. En el punto de aplicación, nodo 1 de la malla de coordenadas (0,5), es la siguiente:

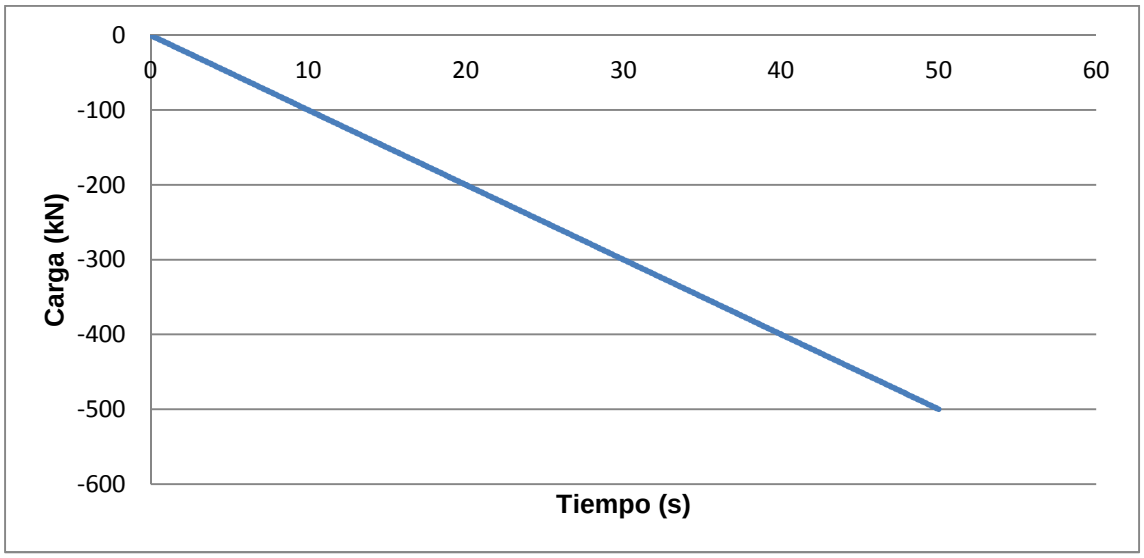

Figura 10. Variación de la carga aplicada respecto del tiempo

Como se puede visualizar, la carga es introducida correctamente en el nodo, partiendo de un valor nulo y aumentando con una pendiente de -10.

También se puede observar como la evolución del axil se realiza de la forma esperada.

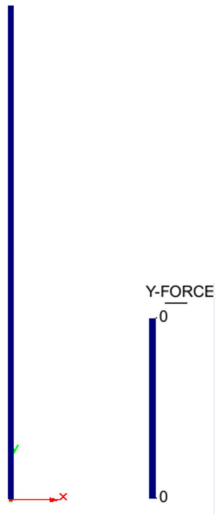

Figura 11. Esfuerzo axil, paso 0

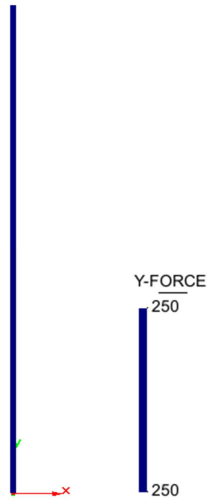

Figura 12. Esfuerzo axil, paso 25

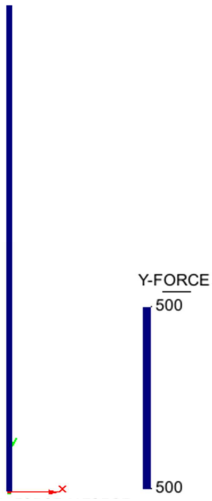

Figura 13. Esfuerzo axil, paso 50

El axil va aumentando a medida que la carga también aumenta. Además, se comprueba la variación de la reacción.

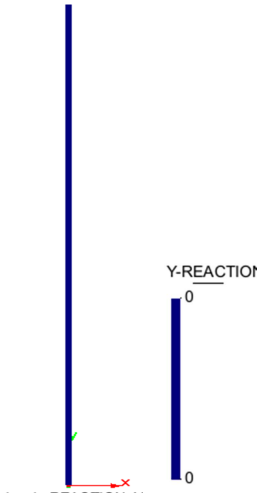

Figura 14. Reacción vertical, paso 0

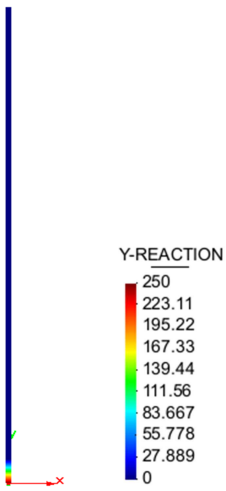

Figura 15. Reacción vertical, paso 25

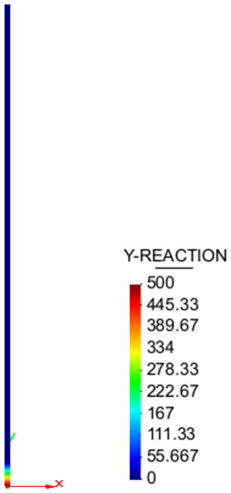

Figura 16. Reacción vertical, paso 50

Por tanto, se concluye que la condición de la condición de contorno variable en el tiempo, la carga puntual, es aplicada correctamente a la estructura.

5.2 PÓRTICO CON UN PILAR SOMETIDO A UN DESPLAZAMIENTO IMPUESTO

5.2.1 Descripción del problema

La estructura de análisis en cuestión consiste en un pórtico, compuesto por dos pilares y una viga, formando un ángulo recto entre ellos y biapoyado. El pórtico mide 5m de altura por 5m de ancho. Los pilares y la viga son de sección cuadrada, ambas de 0.2x0.2m. El material por el que están formados tiene las siguientes características:

Módulo de Elasticidad	$2{,}0 \cdot 10^{11}$	N/m^2
Número de Poisson	0,29	-
Densidad	7870	Kg/m^3

Tabla 2. Características mecánicas

La estructura va a estar sometida a un desplazamiento horizontal impuesto, variable en el tiempo, en el punto de apoyo del pilar derecho. Este desplazamiento será impuesto en formato tabla, con un tiempo total de 1s y con un paso de tiempo de 0,1s. La tabla tiempo – desplazamiento es la siguiente:

Tiempo (s)	Desplazamiento (m)
0	0
0,1	0,05
0,2	0,03
0,3	0,04
0,4	0,02
0,5	0,05
0,6	0,07
0,7	0,01
0,8	0,09
0,9	0,12
1	0,08

Tabla 3. Valores a imponer de tiempo-desplazamiento

El efecto del peso propio se va a despreciar, de forma que el único efecto al que se vea afectada la estructura sea el desplazamiento impuesto.

El esquema estructural del problema es el siguiente:

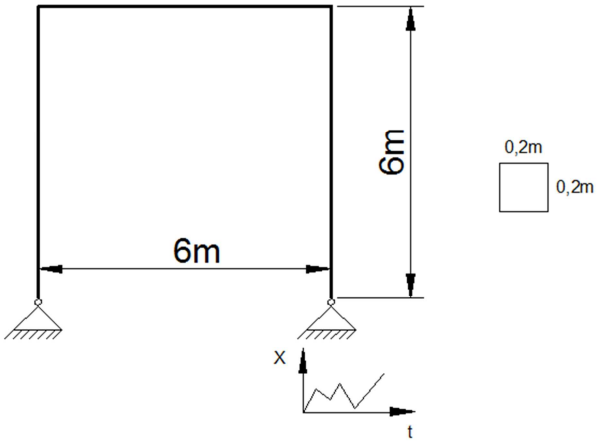

Figura 17. Esquema estructural

El objetivo de este ejemplo es el de validar si la condición de contorno variable en el tiempo, que en este caso es el desplazamiento en el apoyo de un pilar del pórtico, se impone correctamente.

5.2.2 Procedimiento a seguir para la resolución del problema

Primeramente, se introduce la geometría dada en GiD.

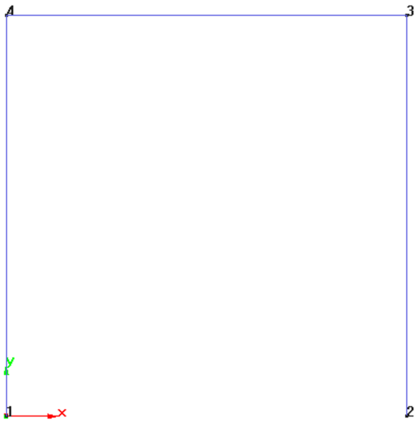

Figura 18. Geometría

Se declara a Kratos como *problemtype*, para así poder aplicar sus condiciones. Una vez inicializado, se define el material de estudio, con las propiedades mecánicas correspondientes.

A continuación, se deben definir las características del modelo.

- Tipo de análisis

El tipo de análisis a realizar es genérico, dinámico, lineal y enfocado a pequeños desplazamientos

- Estrategia de solución

Se define como tiempo total 1s, con un paso de 0,1s. Por tanto, el número total de pasos será de 10.

- Propiedades

En este apartado se escoge el material anteriormente creado, la sección rectangular 0,2x0,2m y el tipo de propiedad, que en este caso es *Beam,* ya que se trabaja con vigas y pilares. Con todas estas características se define una propiedad.

La propiedad creada es la que se asigna a los elementos dibujados. Sólo ha hecho falta crear una debido a que hay una continuidad total entre los pilares i la viga.

- Cargas

Como se desprecia el peso propio, la estructura no es afectada por ninguna carga.

- Condiciones de contorno

Como se ha introducido anteriormente, el pórtico está biapoyado en la base de sus pilares, de forma que los desplazamientos están impedidos, pero sí que están permitidos los giros.

También se debe “transformar” el problema de 3D a 2D. Esto se consigue aplicando, a las líneas que representan los pilares y la viga, desplazamiento libre en X y en Y, y rotación libre alrededor de Z; mientras que el desplazamiento en Z y la rotación en X y en Y deben estar impedidos.

- Deplazamientos impuestos

Para aplicar estos desplazamientos impuestos variables en el tiempo, es necesario utilizar la interfaz creada en Python para poder introducirla, ya que en el cálculo convencional de Kratos no es posible imponer esta condición. Para poder utilizarla, primero se tendrá que realizar un “precálculo”, de forma que se generen los archivos necesarios.

Antes del “precálculo”, pero, se debe generar la malla. En este caso, se trata de una malla no estructurada triangular, compuesta de 16 nodos y de 15 elementos.

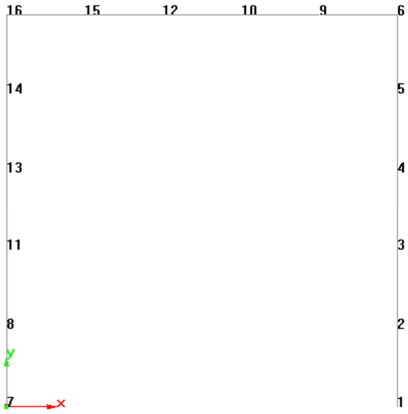

Figura 19. Malla

Después de realizar el mallado ya se puede realizar el “precálculo”. En la carpeta donde se almacenan los ficheros del problema de estudio se obtienen los archivos *KratosStructuralOpenMP y ProjectParameters*. Es sobre estos archivos, como se ha explicado en el capítulo 4, donde se podrán realizar las modificaciones convenientes para así poder introducir los desplazamientos variables en el tiempo.

En este caso, el proceso personalizado (*custom process*) que se crea en el archivo *KratosStructuralOpenMP* es el siguiente:

```
########################################################################
# 1 CREATE CUSTOM PROCESSES
import apply_nodal_variable_process

customProcesses = []
class nodal_variable_settings:
    # these 2 parameters are for the generation of the process
    file_name = "apply_nodal_variable_process"
    class_name = "apply_nodal_variable"
    group_id = 0
    # parameters used by this custom process
    fix = True
    value = None
    time_table =
[[0.0,0.0],[0.1,0.05],[0.2,0.03],[0.3,0.04],[0.4,0.02],[0.5,0.05],
[0.6,0.07],[0.7,0.01],[0.8,0.09],[0.9,0.12],[1.0,0.08]]
    time_begin = 0.0
    time_end = 1
    default_value = 0.0
    keep_after_end = False
    nodes = [1]
    variable_name = "DISPLACEMENT_X"

proc1 = apply_nodal_variable_process.CreateCustomProcess(model_part,
nodal_variable_settings)
```

```
customProcesses.append(proc1)
#######################################################################
```

Como se puede apreciar:

- Se crea la clase *nodal_variable_settings*.
- Se utiliza el proceso *apply_nodal_variable_process* para poder aplicar el desplazamiento impuesto.
- La variable de estudio es llamada “DISPLACEMENT_X”, ya que se trata de un desplazamiento en dirección horizontal.
- Se define el nodo de aplicación, el número 1.
- Se inserta la tabla dada, de tipo tiempo - valor. Cada paso de tiempo tiene su correspondiente desplazamiento horizontal impuesto.
- Se establecen el tiempo de inicio y final de la aplicación de la condición, de 0 a 1 segundo.

Con esta modificación, junto con las otras generales, ya se puede ejecutar el runkratos.exe y realizar el cálculo del problema.

Una vez realizado, accediendo al apartado de postproceso de GiD, ya se pueden ver los resultados obtenidos.

5.2.3 Análisis de los resultados obtenidos

El proceso de validación consistirá en analizar cuál es el movimiento horizontal que experimenta el nodo 1 (punto base del pilar derecho del pórtico). Se deberá comprobar que el desplazamiento impuesto en este nodo coincida con los datos de la tabla.

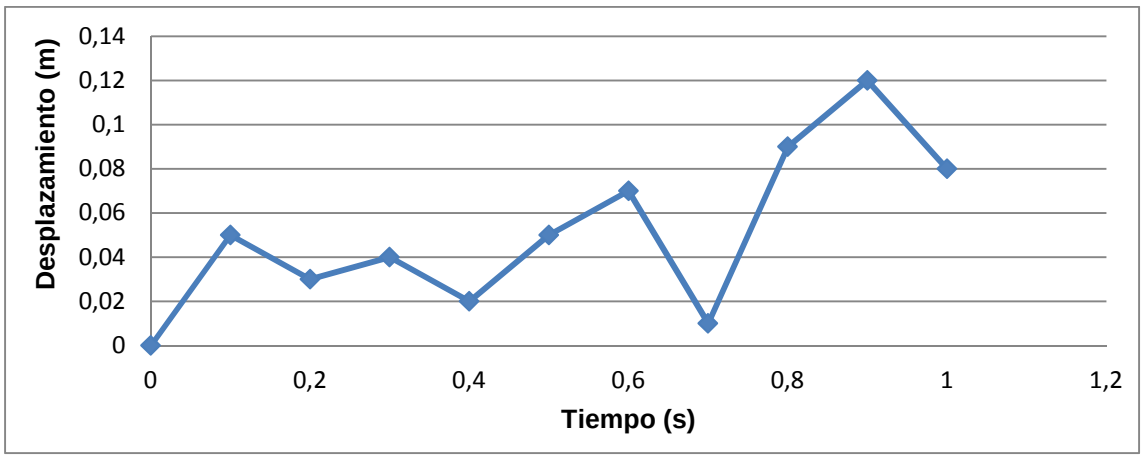

Figura 20. Gráfica de la variación del desplazamiento impuesto en el nodo 1

Los datos de la gráfica coinciden con los de la tabla tiempo-desplazamiento. Por tanto, se puede confirmar que la implementación de la condición de contorno ha sido satisfactoria.

A continuación, se procede a ver cuál es la respuesta de la estructura al desplazamiento variable impuesto. Se presentan algunos pasos, en los que se puede observar el desplazamiento horizontal global y su deformada, aumentada 100 veces.

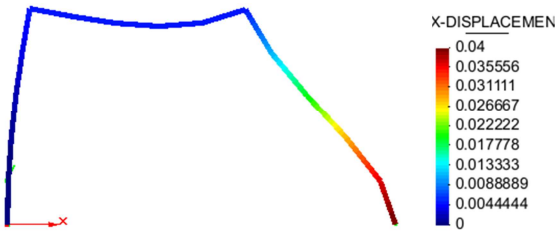

Figura 21. Desplazamiento horizontal, paso 3

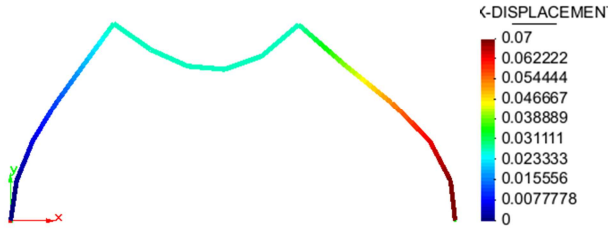

Figura 22. Desplazamiento horizontal, paso 6

Figura 23. Desplazamiento horizontal, paso 9

5.3 PLACA SOMETIDA A UNA CARGA EN RAMPA

Una vez realizados estos dos primeros ejemplos sobre elementos tipo *Beam* (viga) y comprobando que la correcta aplicación de las condiciones de contorno, se procede a realizar el análisis sobre el elemento tipo *Shell* (placa).

5.3.1 Descripción del problema

La estructura de análisis consiste en una placa circular empotrada en su contorno exterior, de 6,4m de radio y con un espesor de 0,76m. Las características del material son las siguientes:

Módulo de Elasticidad	$20{,}0 \cdot 10^9$	N/m^2
Número de Poisson	0,27	-
Densidad	2549,29	Kg/m^3

Tabla 4. Características mecánicas

Esta placa va a estar sometido a las cargas de su peso propio y, además, a una carga puntual en rampa, es decir, de magnitud variable en el tiempo. Su variación respecto del tiempo viene dado en la gráfica siguiente:

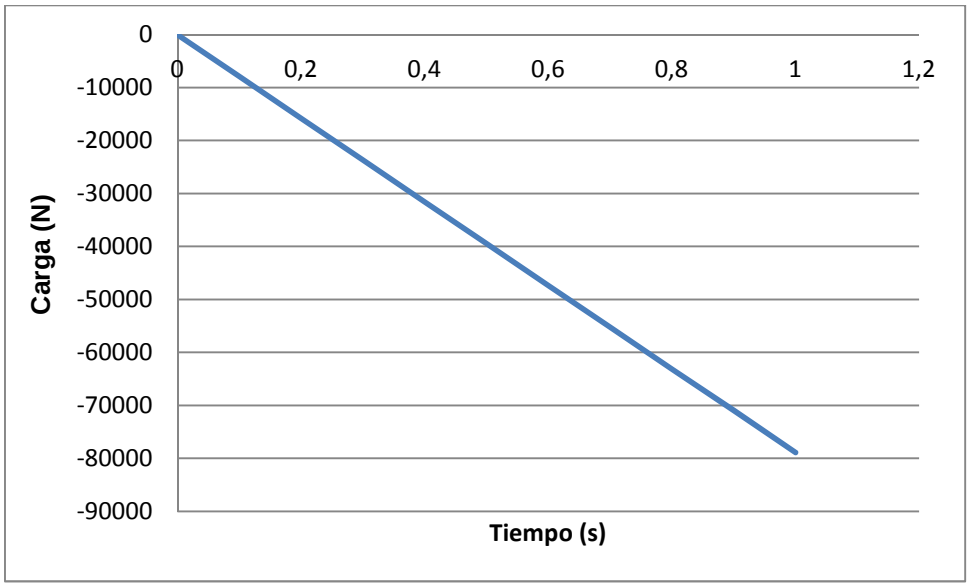

Figura 24. Gráfica de la evolución de la carga con el tiempo

La carga va aumentando gradualmente, con una pendiente de -78900. El punto de aplicación de la carga puntual va a ser el centro de la placa.

El esquema estructural de la estructura a resolver es el siguiente:

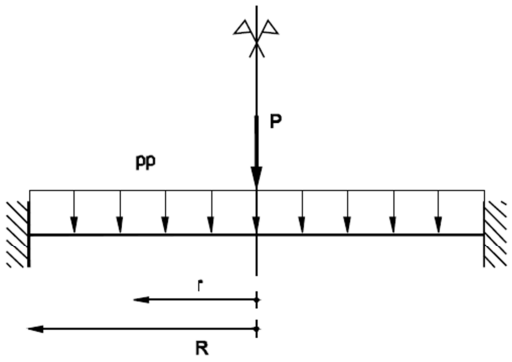

Figura 25. Esquema estructural

El objetivo de este ejemplo es el de analizar el desplazamiento en el punto de aplicación de la carga, el punto central de la placa. El resultado obtenido se va a comparar con la solución teórica calculada.

5.3.2 Procedimiento a seguir para la resolución del problema

Primero de todo, se introduce la geometría dada en GiD.

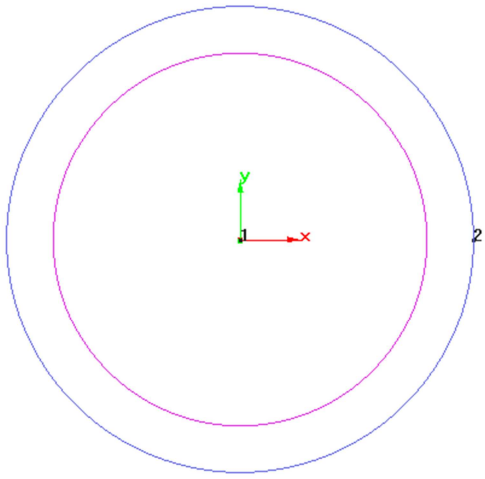

Figura 26. Geometría

Declarando a Kratos como *problemtype*, se pueden aplicar sus condiciones. Una vez inicializado, se define el material de estudio, con las propiedades mecánicas anteriormente descritas.

A continuación, se deben definir las características del modelo.

- Tipo de análisis

El tipo de análisis a realizar es genérico, estático, lineal y enfocado a pequeños desplazamientos.

- Estrategia de solución

Se define como tiempo total de 1s, con un paso de 0,1s. El total de pasos van a ser 10.

- Propiedades

En este apartado se escoge el material anteriormente creado y el tipo de propiedad, que en este caso es *shell* (porque se trabaja con una placa), con el espesor correspondiente de 0,76m. Con todas estas características se define una propiedad.

La propiedad creada es la que se asigna al elemento dibujado.

- Condiciones de contorno

Como se ha mencionado en el planteamiento del problema, la estructura está empotrada en todo su contorno, por lo que en este no se van a permitir ni los desplazamientos ni las rotaciones en cualquier dirección.

- Cargas

Por un lado, tenemos el peso propio, de valor 19000N/m^2. Esta carga se ha aplicado como una carga repartida por unidad de superficie (*surface load*) ya que hasta el momento Kratos no introduce correctamente las cargas de peso propio correspondientes al caso de elementos tipo *shell*.

Por otro lado, tenemos a las cargas en rampa aplicada en el centro, variable hasta 1s. Este tipo de condición no es una función convencional que realice Kratos, por lo que va a ser necesario utilizar la interfaz creada en Python para poder introducirla. Para poder utilizarla, primero se tendrá que realizar un “precálculo”, donde sólo va a ser el peso propio la carga actuante, para que se generen los archivos necesarios y así introducir las modificaciones necesarias.

Antes de realizar el “precálculo” se debe generar el mallado del elemento. En este caso se trata de una malla no estructurada triangular, compuesta de 269 nodos y de 456 elementos. Se ha realizado una malla más fina con el objetivo de obtener unos resultados precisos aptos para ser comparables.

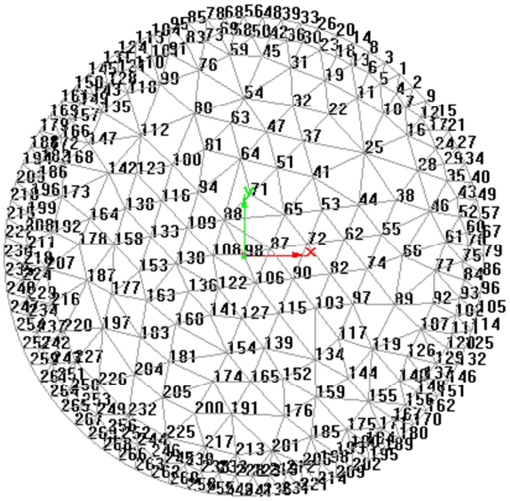

Figura 27. Malla

Ahora sí que se puede realizar el “precálculo”. En la carpeta donde se almacenan los ficheros del problema de estudio veremos que se obtienen los archivos *KratosStructuralOpenMP y ProjectParameters*. Es sobre estos archivos, como se ha explicado en el capítulo 4, donde se podrán realizar las modificaciones convenientes para así poder introducir las cargas variables en el tiempo del problema.

En este caso, el proceso personalizado (*custom process*) que se crea en el archivo *KratosStructuralOpenMP* es el siguiente:

```
#######################################################################
# 1 CREATE CUSTOM PROCESSES
import apply_nodal_variable_process

def user_time_function_proc1(t):
    return -t*(78900)

customProcesses = []
class nodal_variable_settings:
    # these 2 parameters are for the generation of the process
    file_name = "apply_nodal_variable_process"
    class_name = "apply_nodal_variable"
    group_id = 0
    # parameters used by this custom process
    fix = True
    value = None
    time_table = None
    user_time_function = user_time_function_proc1
    time_begin = 0.0
    time_end = 1.0
    default_value = 0.0
    keep_after_end = True
    nodes = [98] # this is temporary. later on we'll use the group_id
    variable_name = "POINT_LOAD_Z"
```

```
proc1 = apply_nodal_variable_process.CreateCustomProcess(model_part,
nodal_variable_settings)
customProcesses.append(proc1)

#######################################################################
```

Como se puede ver:

- Se crea la clase *nodal_variable_settings*, donde se definen todos los parámetros necesarios.
- Se utiliza el proceso *apply_nodal_variable_process*.
- La variable de estudio es llamada “POINT_LOAD_Z”, debido a que la carga es puntual.
- Se definen el nodo central de aplicación, el 98.
- La aplicación de la carga en rampa viene definida por la función user_time_function_proc1.
- Se establecen el tiempo de inicio y final de la aplicación de la carga.

Con esta modificación, junto con las otras generales, ya se puede ejecutar el runkratos.exe y realizar el cálculo del problema.

Una vez realizado, accediendo al apartado de postproceso de GiD, ya se pueden ver los resultados obtenidos.

5.3.3 Análisis de los resultados obtenidos

- Los resultados obtenidos con Kratos, correspondientes al desplazamiento Z en el último paso, son los siguientes:

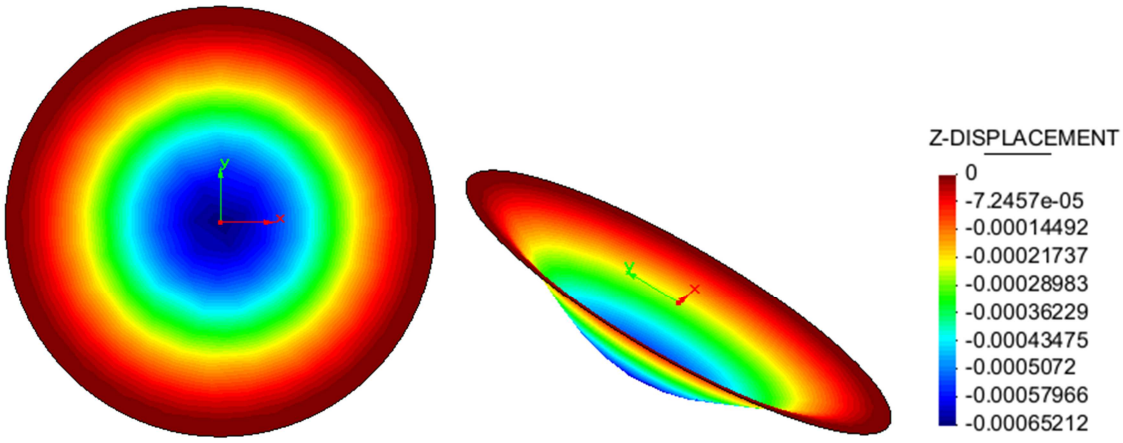

Figura 28. Desplazamiento global de la placa, paso 1. Deformación x 3500

El valor de la flecha total correspondiente al instante de tiempo 1, bajo el efecto de la carga puntual de valor 78900N y el peso propio, es de **6,52·10^{-4}m**. Este es el valor que se comprobará con el teórico.

- El cálculo teórico se realiza se procede de la siguiente forma:

Se tiene que calcular de la expresión general de la flecha, partiendo de la siguiente ecuación diferencial:

$$Q_r = -D \cdot \frac{d}{dr}\left[\frac{1}{r} \cdot \frac{d}{dr}\left(r\frac{d\omega}{dr}\right)\right]$$

Donde Q_r corresponde a la expresión del cortante, la cual se determina por equilibrio.

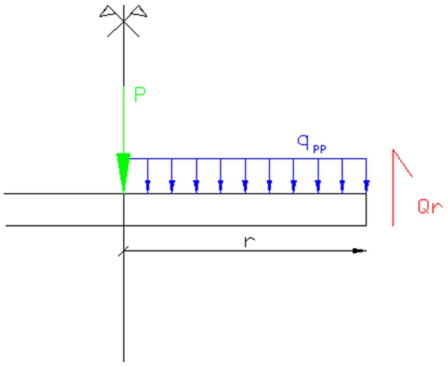

Figura 29. Equilibrio de fuerzas de la placa

$$Q_r \cdot 2\pi r = P + \pi r^2 q_{p_p}$$

$$Q_r = -\left(\frac{P + \pi r^2 q_{p_p}}{2\pi r}\right) = -\left(\frac{78.9 + 19\pi r^2}{2\pi r}\right)$$

donde r es la posición a lo largo del radio.

Sustituyendo la expresión del cortante encontrada:

$$\frac{d}{dr}\left[\frac{1}{r} \cdot \frac{d}{dr}\left(r\frac{d\omega}{dr}\right)\right] = \frac{P + \pi r^2 q_{p_p}}{2D\pi r}$$

A continuación, para obtener la expresión de la flecha debemos integrar tres veces la ecuación diferencial:

- Primera integración:

$$\int \frac{d}{dr}\left[\frac{1}{r}\cdot\frac{d}{dr}\left(r\frac{d\omega}{dr}\right)\right]dr = \int \frac{P+\pi r^2 q_{p_p}}{2D\pi r}dr$$

$$\frac{d}{dr}\left(r\frac{d\omega}{dr}\right) = \frac{P\cdot ln(r)\cdot r}{2D\pi} + \frac{q_{p_p}r^3}{4D} + C_1 r$$

- Segunda integración:

$$\int \frac{d}{dr}\left(r\frac{d\omega}{dr}\right)dr = \int \left(\frac{P\cdot ln(r)\cdot r}{2D\pi} + \frac{q_{p_p}r^3}{4D} + C_1 r\right)dr$$

$$\frac{d\omega}{dr} = \frac{Pr}{4D\pi}\left(ln(r)-\frac{1}{2}\right) + \frac{q_{p_p}r^3}{16D} + \frac{C_1 r}{2} + \frac{C_2}{r}$$

- Tercera integración:

$$\int \frac{d\omega}{dr}dr = \int \left(\frac{Pr}{4D\pi}\left(ln(r)-\frac{1}{2}\right) + \frac{q_{p_p}r^3}{16D} + \frac{C_1 r}{2} + \frac{C_2}{r}\right)dr$$

$$\omega = \frac{Pr^2}{8D\pi}(ln(r)-1) + \frac{q_{p_p}r^4}{64D} + \frac{C_1 r^2}{4} + C_2 ln(r) + C_3$$

Como se puede apreciar, aparecen tres constantes de integración, que se encuentran aplicando las condiciones de contorno de empotramiento:

$$(1)\ \frac{d\omega}{dr}_{r=0} = 0 \ \rightarrow\ C_2 = 0$$

$$(2)\ \frac{d\omega}{dr}_{r=R} = 0 \ \rightarrow\ \frac{PR}{4D\pi}\left(ln(R)-\frac{1}{2}\right) + \frac{q_{p_p}R^3}{16D} + \frac{C_1 R}{2} + \frac{C_2}{R} = 0$$

$$C_1 = -\left[\frac{P}{2D\pi}\left(ln(R)-\frac{1}{2}\right) + \frac{q_{p_p}R^2}{8D}\right]$$

$$(3)\ \omega_{r=R} = 0 \ \rightarrow\ \frac{PR^2}{8D\pi}(ln(R)-1) + \frac{q_{p_p}R^4}{64D} + \frac{C_1 R^2}{4} + C_3 = 0$$

sustituyendo C1 se obtiene:

$$C_3 = \frac{PR^2}{16D\pi} + \frac{q_{p_p}R^4}{64D}$$

Una vez encontradas las constantes de integración, se sustituyen en las ecuaciones encontradas anteriormente:

$$\omega = \frac{Pr^2}{8D\pi}(ln(r) - 1) + \frac{q_{p_p}r^4}{64D} - \frac{r^2}{4}\left[\frac{P}{2D\pi}\left(ln(R) - \frac{1}{2}\right) + \frac{q_{p_p}R^2}{8D}\right] + \frac{PR^2}{16D\pi} + \frac{q_{p_p}R^4}{64D}$$

donde D se calcula de la siguiente forma:

$$D = \frac{Et^3}{12(1-v^2)} = \frac{22000000 \cdot 0.76^3}{12(1-0.27^2)} = 868071.77\ kNm$$

Así que la expresión de la flecha para cualquier punto de la placa es la siguiente:

$$\omega = \frac{78.9r^2}{8D\pi}(ln(r) - 1) + \frac{19r^4}{64D} - \frac{r^2}{4}\left[\frac{53.51}{D\pi} + \frac{97.28}{D}\right] + \frac{201.98}{D\pi} + \frac{498.07}{D}$$

Particularizando el valor de la flecha en el centro, con $r = 0$, se obtiene la flecha es de valor **6,48·10^{-4}m**.

Por tanto, comparando los resultados obtenidos, se observa que son prácticamente los mismos. La diferencia entre ellos es del 0,6%. Se concluye que el análisis ha sido realizado correctamente.

5.4 ESTRUCTURA SOMETIDA A CARGAS PUNTUALES VARIABLES

El ejemplo utilizado a continuación ha sido seleccionado de un manual de validación de XFINAS [10].

XFINAS [9] es un programa de elementos finitos no lineal, que es capaz de evaluar la estabilidad de las estructuras de pared delgada y permite un potente análisis dinámico estructural no lineal del material y de la geometría.

5.4.1 Descripción del problema

La estructura de análisis en cuestión consiste en un pilar y una viga, formando un ángulo recto entre los dos. El pilar y la viga tienen unas longitudes de 3m y 2m respectivamente, ambos con una sección cuadrada de 0,1m y apoyados en sus dos extremos. En el nodo 1 sólo se permiten los giros, mientras que en el 13 además de permitirse los giros también se permite el desplazamiento horizontal.

Módulo de Elasticidad	$2{,}0 \cdot 10^{11}$	N/m^2
Número de Poisson	0,29	-
Densidad	7860	Kg/m^3

Tabla 5. Características mecánicas

Ésta estructura va a estar sometida a varias cargas arbitrarias y puntuales P, de valor:

$$P_i(t) = \sin(100t)\,, i = 1{,}2{,}3$$

donde t es el tiempo.

Estas cargas son variables harmónicas en el tiempo, por lo que se deberán tener en cuenta en el proceso de Python creado.

Se va a despreciar el efecto del peso propio de la estructura, con la finalidad de ver directamente cual es el efecto de las cargas aplicadas arbitrariamente.

El esquema estructural de la estructura a resolver es el siguiente:

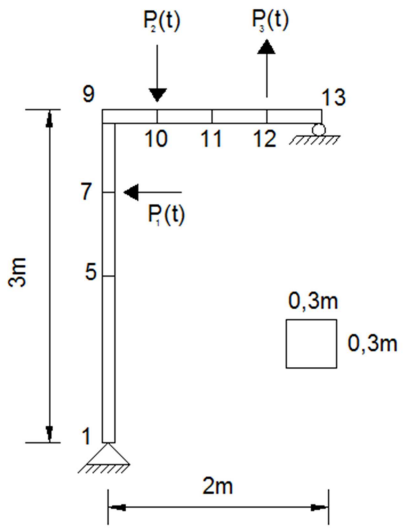

Figura 30. Esquema estructural

El objetivo del problema va a ser el de estudiar cuál es el desplazamiento horizontal a lo largo del tiempo al cual se ve afectado el punto medio del pilar, el nodo 5. La gráfica que se obtenga es la que se va a comparar con la obtenida mediante el programa Xfinas, información obtenida del manual de validación del programa.

5.4.2 Procedimiento a seguir para la resolución del problema

Primero de todo, se introduce la geometría dada en GiD.

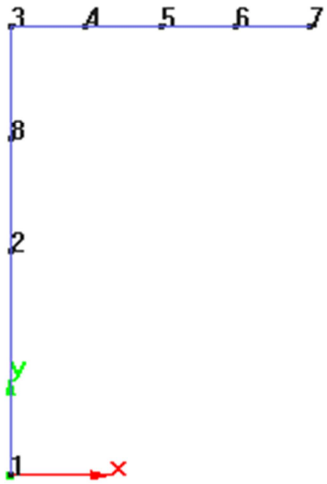

Figura 31. Geometría

Se declara a Kratos como *problemtype*, para así poder aplicar sus condiciones. Una vez inicializado, se define el material de estudio, con las propiedades mecánicas anteriormente descritas.

A continuación, se deben definir las características del modelo.

- Tipo de análisis

  El tipo de análisis a realizar es genérico, dinámico, no lineal y enfocado a pequeños desplazamientos.

- Estrategia de solución

  Se define como tiempo total 1s, con un paso de 0,0025s.

- Propiedades

En este apartado se escoge el material anteriormente creado, la sección rectangular 0,1x0,1m y el tipo de propiedad, que en este caso es *Beam,* ya que se trabaja con elementos tipo viga y pilar. Con todas estas características se define una propiedad.

La propiedad creada es la que se asigna a los elementos dibujados. Sólo ha hecho falta crear una debido a que hay una continuidad total entre el pilar i la viga.

- Condiciones de contorno

En el nodo 1, los desplazamientos son impedidos en cualquier dirección, mientras que en el nodo 13 sólo es impedido el desplazamiento Y, mientras que el desplazamiento X no está impedido. Ambos apoyos permiten los giros alrededor de los ejes X e Y.

También se debe "transformar" el problema de 3D a 2D. Esto se consigue aplicando, a las líneas que representan el pilar y la viga, desplazamiento libre en X y en Y, y rotación libre alrededor de Z. El desplazamiento en Z, y la rotación en X y en Y deben estar impedidos.

- Cargas

La estructura se ve afectada por las cargas puntuales P que varían en función del tiempo. Este tipo de condición no es una función convencional que realice Kratos, por lo que va a ser necesario utilizar la interfaz creada en Python para poder introducirla. Para hacerlo, primero se tendrá que realizar un "precálculo", de forma que se generen los archivos Python en la carpeta del problema, sobre las cuales se van a realizar las correspondientes modificaciones.

Aun así, se debe crear la condición inicial del efecto de estas cargas puntuales en sus nodos correspondientes, ya que se necesita que Kratos las genere en el archivo .mdpa.

Antes de proceder al "precálculo", pero, se debe generar el mallado del elemento. En este caso se trata de una malla no estructurada triangular, compuesta de 13 nodos y de 12 elementos.

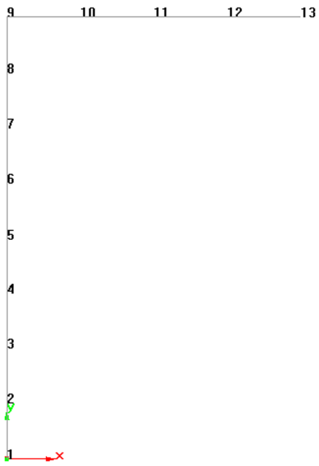

Figura 32. Malla

Con la malla creada, ya se puede realizar el "precálculo". En la carpeta donde se almacenan los ficheros del problema de estudio veremos que se obtienen los archivos *KratosStructuralOpenMP y ProjectParameters*. Es sobre estos archivos, como se ha explicado en el capítulo 4, donde se podrán realizar las modificaciones convenientes para así poder introducir las cargas variables en el tiempo del problema.

En este caso, el proceso personalizado (*custom process*) que se crea en el archivo *KratosStructuralOpenMP* es el siguiente:

```
#######################################################################
1 CREATE CUSTOM PROCESSES
import apply_nodal_variable_process
import math

def user_time_function_proc1(t):
        return math.sin(100*t)
def user_time_function_proc2(t):
        return -math.sin(100*t)

customProcesses = []

class nodal_variable_settings1:
    # these 2 parameters are for the generation of the process
    file_name = "apply_nodal_variable_process"
    class_name = "apply_nodal_variable"
    group_id = 0
    # parameters used by this custom process
    fix = True
    user_time_function = user_time_function_proc1
    value = None
    user_spatial_function=None
    time_table = None
    time_begin = 0.0
    time_end = 1.0
    default_value = 0.0
    keep_after_end = False
```

```
    nodes = [12]
    variable_name = "POINT_LOAD_Y"

proc1 = apply_nodal_variable_process.CreateCustomProcess(model_part,
nodal_variable_settings1)
customProcesses.append(proc1)

class nodal_variable_settings2:
    # these 2 parameters are for the generation of the process
    file_name = "apply_nodal_variable_process"
    class_name = "apply_nodal_variable"
    group_id = 0
    # parameters used by this custom process
    fix = True
    user_time_function = user_time_function_proc2
    value = None
    user_spatial_function=None
    time_table = None
    time_begin = 0.0
    time_end = 1.0
    default_value = 0.0
    keep_after_end = False
    nodes = [10]
    variable_name = "POINT_LOAD_Y"

proc2 = apply_nodal_variable_process.CreateCustomProcess(model_part,
nodal_variable_settings2)
customProcesses.append(proc2)

class nodal_variable_settings3:
    # these 2 parameters are for the generation of the process
    file_name = "apply_nodal_variable_process"
    class_name = "apply_nodal_variable"
    group_id = 0
    # parameters used by this custom process
    fix = True
    user_time_function = user_time_function_proc2
    value = None
    user_spatial_function=None
    time_table = None
    time_begin = 0.0
    time_end = 1.0
    default_value = 0.0
    keep_after_end = False
    nodes = [7]
    variable_name = "POINT_LOAD_X"

proc3 = apply_nodal_variable_process.CreateCustomProcess(model_part,
nodal_variable_settings3)
customProcesses.append(proc3)

#######################################################################
```

Como se puede ver:

- Se crean tres clases distintas, una por cada carga a aplicar.
- Se utiliza el proceso *apply_nodal_variable_process*.

- La variable de estudio es llamada “POINT_LOAD_X” o “POINT_LOAD_Y” en función de cuál sea la dirección de la carga puntual a aplicar.
- Se definen los nodos de aplicación: el 7, el 10 y el 12.
- Se definen las funciones P(t), llamadas *user_time_function_proc1* y *user_time_function_proc2,* como dato de entrada. Cada clase usa su correspondiente.
- Se establecen el tiempo de inicio y final de aplicación de las cargas.

Con esta modificación, junto con las otras generales, ya se puede ejecutar el runkratos.exe y realizar el cálculo del problema.

Una vez realizado, accediendo al apartado de postproceso de GiD, ya se pueden ver los resultados obtenidos.

5.4.3 Análisis de los resultados obtenidos

Inicialmente, se comprueba que las cargas aplicadas han sido definidas correctamente.

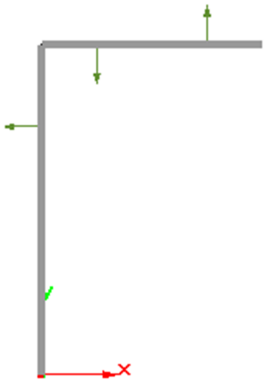

Figura 33. Puntos y dirección de aplicación de las cargas

Además, también se comprueba que la evolución de su valor a lo largo del tiempo sea correcta.

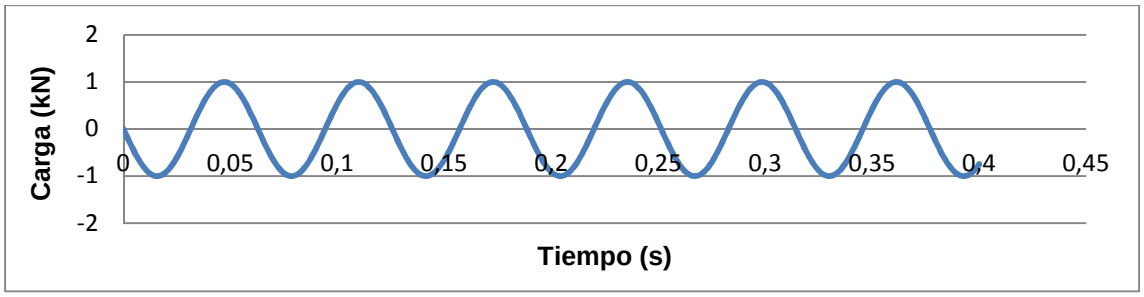

Figura 34. Gráfica correspondiente a la variación en el tiempo de la carga 1

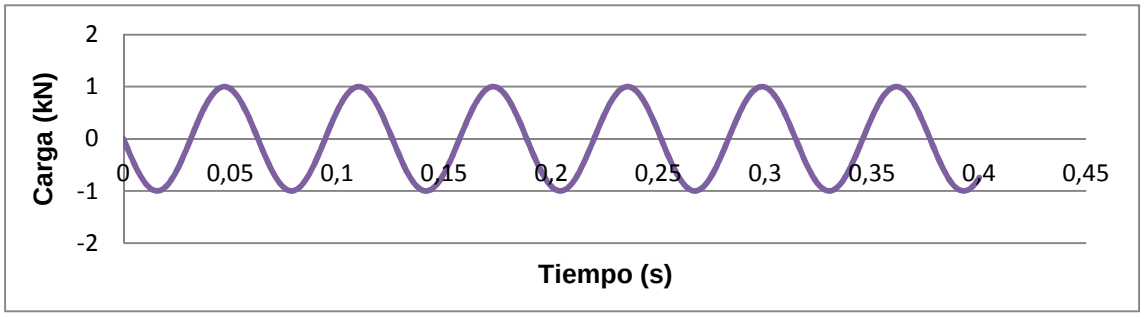

Figura 35. Gráfica correspondiente a la variación en el tiempo de la carga 2

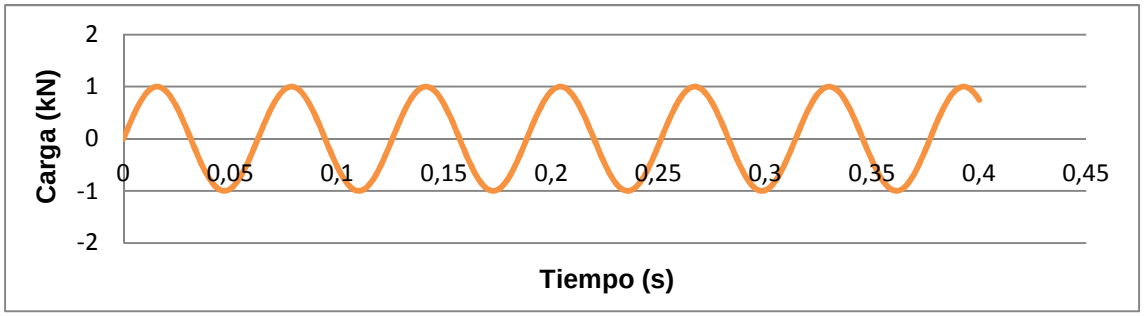

Figura 36. Gráfica correspondiente a la variación en el tiempo de la carga 3

Como se puede observar, las cargas han sido aplicadas correctamente.

Una vez verificado la correcta introducción de las condiciones, se observa el desplazamiento global de la estructura. Para visualizar de mejor forma los resultados, la deformación ha sido aumentada 50000 veces.

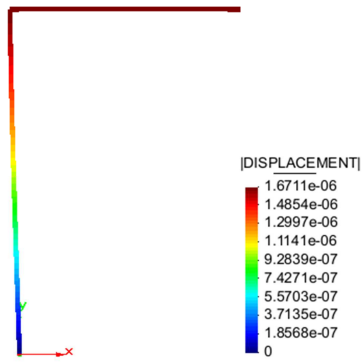

Figura 37. Deformación x 50000, paso 0,06

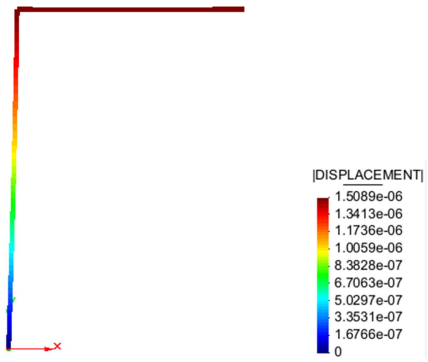

Figura 38. Deformación x 50000, paso 0,265

Se procede a evaluar el desplazamiento horizontal del nodo 5, situado en el centro del pilar, para así poder compararlo con los resultados obtenidos con el programa XFINAS.

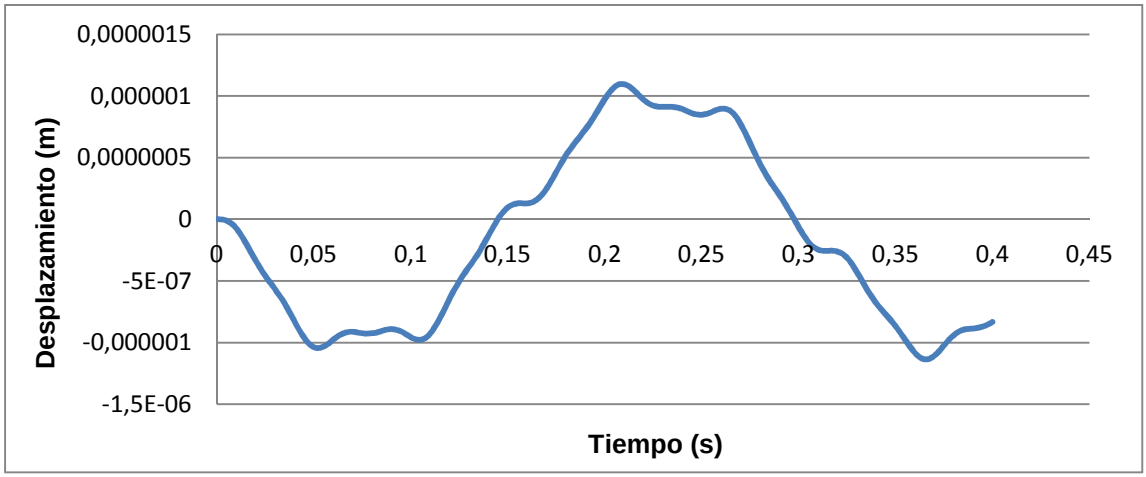

Figura 39. Gráfica del movimiento horizontal del nodo 5 obtenida mediante la interfaz creada

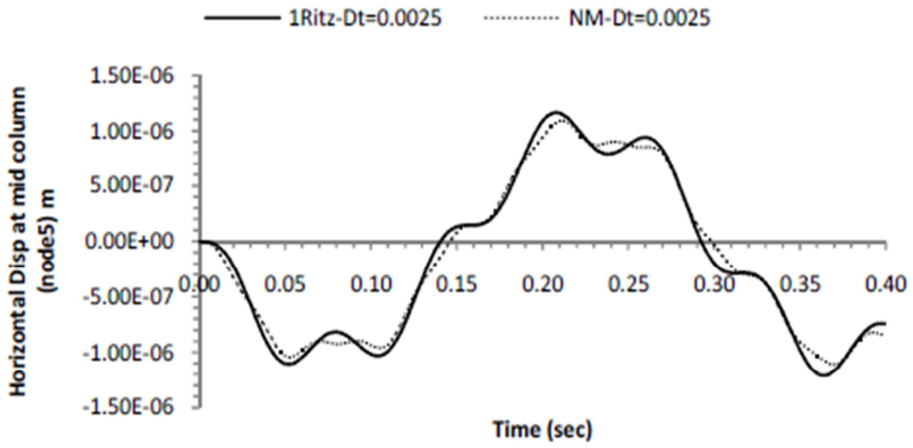

Figura 40. Gráfica del movimiento horizontal del nodo 5 obtenida mediante el programa XFINAS

La solución es calculada mediante el método de Newton-Raphson, por tanto, la comparación se debe realizar con la curva que realiza este método (línea de puntos).

Cualitativamente, se puede observar que las variaciones en el desplazamiento son las mismas. Las pequeñas variaciones entre las dos gráficas son debidas a que los programas usan estrategias de solución distintas, ya sea referente a como se realiza la aplicación de las cargas sobre los elementos, o debido a diferencias entre las estrategias de tiempo. Ésta parte está fuera del alcance de este trabajo, por tanto, se concluye que el ejemplo se ha realizado correctamente.

5.5 PÓRTICO SOMETIDO A ACCIONES SÍSMICAS

Los ejemplos que se presentan a continuación han sido seleccionados de un listado de OpenSees [8].

OpenSees (*Open System for Earthquake Engineering Simulation*) es un *software* utilizado para investigación y simulación de sistemas geotécnicos y estructurales. Ha sido desarrollado por el *Pacific Eartquake Engineering Research Center* (PEER) con el apoyo del *National Sciene Foundation*.

En los dos ejemplos que se presentan a continuación, se les van a aplicar unas condiciones de contorno sísmicas. Éstas se pueden aplicar como un desplazamiento, una velocidad o una aceleración [7]. En este caso, a la estructura de análisis, en el primer ejemplo se le va a aplicar un sismo con el desplazamiento horizontal como variable; y en el segundo se va a ser la aceleración horizontal la variable a introducir.

En este apartado, la parte inicial correspondiente a la descripción y construcción del problema se va a realizar de forma conjunta para ambos ejemplos, ya que no es hasta el final, cuando se deben realizar las modificaciones en los archivos correspondientes, donde se tendrán que definir las condiciones de contorno.

5.5.1 Descripción del problema

La estructura de análisis en cuestión consiste en un pórtico, compuesto por dos pilares y una viga, formando un ángulo recto entre ellos y biapoyado. El pórtico mide 6m de altura por 9m de ancho. Los pilares y la viga son de sección rectangular, pero de dimensiones 0,5x0,8m y 0,5x0,5m, respectivamente. El material por el que están formados tiene las siguientes características:

Módulo de Elasticidad	$2{,}0 \cdot 10^{11}$	N/m^2
Número de Poisson	0,29	-
Densidad	6526,184	Kg/m^3

Tabla 6. Características mecánicas

Esta estructura va a estar sometida a las acciones su peso propio y a un sismo. Este sismo está representado mediante un desplazamiento o a una aceleración horizontales, impuestos en las bases de los dos apoyos y variables con el tiempo

Los datos de la variación de la condición de contorno del sismo vienen representados por dos archivos en formato lista, ambos obtenidos de la web de OpenSees donde se plantean los ejemplos, como se ha citado anteriormente. El archivo "H-E12140.DT2" tiene como variable el desplazamiento, mientras que el archivo "H-E12140.AT2" tiene como variable la aceleración.

El esquema estructural de la estructura a resolver es el siguiente:

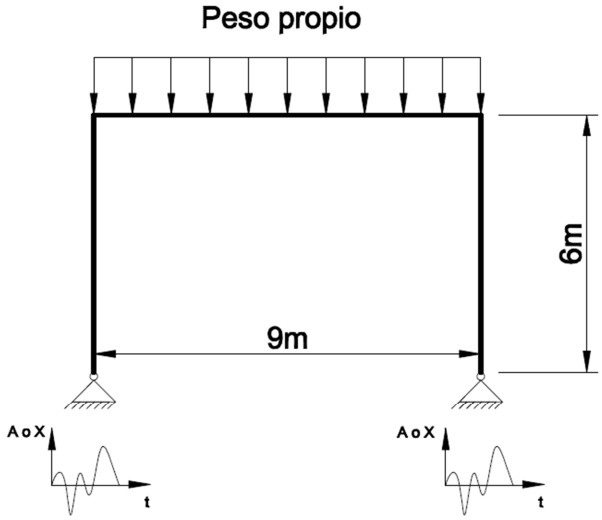

Figura 41. Esquemas estructurales

El objetivo de este análisis es el de comparar los resultados obtenidos con los del programa OpenSees, y comprobar que el proceso *ground_motion_process* aplica las condiciones de contorno variables con el tiempo correctamente.

5.5.2 Procedimiento a seguir para la resolución del problema

Primeramente, se introduce la geometría dada en GiD.

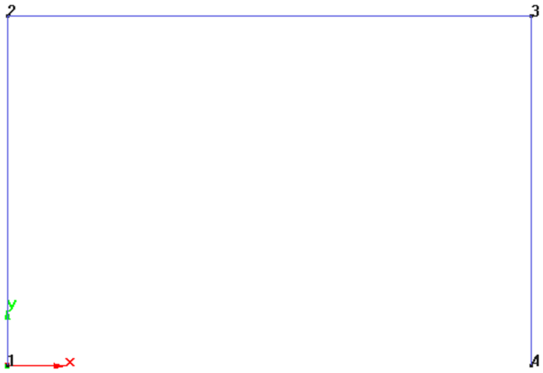

Figura 42. Geometría

Se declara a Kratos como *problemtype*, para así poder aplicar sus condiciones al modelo. Una vez inicializado, se define el material de estudio, con las propiedades mecánicas anteriormente descritas.

A continuación, se deben definir las características del modelo.

- Tipo de análisis

El tipo de análisis a realizar es genérico, dinámico (los desplazamientos impuestos por el sismo varían con el tiempo), lineal y enfocado a pequeños desplazamientos.

- Estrategia de solución

Se define como tiempo total 40s. El paso de tiempo es de 0,01s.

- Propiedades

En este apartado se escoge el material anteriormente creado, la sección rectangular y el tipo de propiedad, que en este caso es *Beam,* ya que se trabaja con vigas y pilares. Hay que definir una propiedad diferente para los pilares y otra para las vigas, debido a que tienen geometrías de sección distintas.

- Cargas

La carga actuante es el peso propio. Esta carga se en el apartado *self-weight*, y se aplica su afecto a toda la estructura.

- Condiciones de contorno

Como se ha mencionado en el planteamiento del problema, la estructura está biapoyada, por lo que en sus extremos no se permiten los desplazamientos (a no

ser que sean desplazamientos impuestos), pero sí que se permiten los giros alrededor el eje X y el eje Y.

También se debe "transformar" el problema de 3D a 2D. Esto se consigue aplicando, a las líneas que representan el pilar y los pórticos, desplazamiento libre en X y en Y, y rotación libre alrededor de Z. El desplazamiento en Z, y la rotación en X y en Y deben estar impedidos.

Por otro lado, tenemos el listado de desplazamientos y de aceleraciones impuestos a los puntos de apoyo, que representan el efecto del sismo sobre los puntos de apoyo de la estructura. Este tipo de condiciones no son una función convencional que realice Kratos, por lo que va a ser necesario utilizar la interfaz creada en Python para poder introducirlas. Para poder utilizarla, primero se tendrá que realizar un "precálculo", donde sólo va a ser el peso propio la carga actuante.

Antes de realizar el "precálculo" se debe generar el mallado del elemento. En este caso se trata de una malla no estructurada triangular, compuesta de 22 nodos y de 21 elementos.

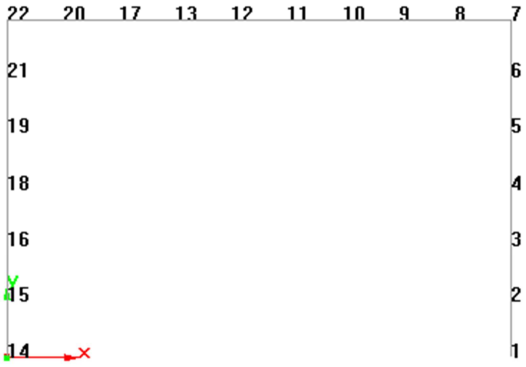

Figura 43. Malla

Ahora sí que se puede realizar el "precálculo". En la carpeta donde se almacenan los ficheros del problema de estudio veremos que se obtienen los archivos *KratosStructuralOpenMP y ProjectParameters*. Es sobre estos archivos, como se ha explicado en el capítulo 4, donde se podrán realizar las modificaciones convenientes para así poder introducir las cargas variables en el tiempo del problema.

Es a partir de aquí donde se debe distinguir entre la elaboración de un ejemplo u otro.

5.5.2.1 Imposición del desplazamiento horizontal como condición de contorno sísmica

El proceso personalizado (*custom process*) que se crea en el archivo *KratosStructuralOpenMP* es el siguiente:

```
########################################################################
# 1 CREATE CUSTOM PROCESSES
import apply_nodal_variable_process
import convert_peer_motion_to_table

disp_record = convert_peer_motion_to_table.convert('H-E12140.DT2')
print('KRATOS: ', disp_record.__class__.__name__)
customProcesses = []
class nodal_variable_settings:
    # these 2 parameters are for the generation of the process
    file_name = "apply_nodal_variable_process"
    class_name = "apply_nodal_variable"
    group_id = 0
    # parameters used by this custom process
    fix = True
    value = None
    time_table = disp_record,
    time_begin = 0.0
    time_end = 39.0
    default_value = 0.0
    keep_after_end = False
    nodes = [1,14]
    variable_name = "DISPLACEMENT_X"

proc1 = apply_nodal_variable_process.CreateCustomProcess(model_part,
nodal_variable_settings)
customProcesses.append(proc1)
########################################################################
```

Como se puede observar:

- Se crea la clase *nodal_variable_settings*, donde se definen todos los parámetros necesarios.
- Se utiliza el proceso *apply_nodal_variable_process*.
- Los desplazamientos impuestos como condición de contorno vienen dados en el archivo 'H-E12140.DT2'. No son dados en un formato de entrada tipo tabla (tiempo – valor) como es requerido, sino que son dados en forma de lista, espaciados entre ellos y cada dato separado por un tiempo de 0.005s. La función *convert_peer_motion_to_table* convierte esta lista de datos en una tabla tiempo – desplazamiento, apta para ser usada por el proceso.
- La variable de estudio es llamada "DISPLACEMENT_X", ya que datos proporcionados son el desplazamiento en esta dirección.

- Se definen los nodos a los que se les aplicarán estas condiciones de contorno, el 1 y el 14.
- Se establecen el tiempo de inicio y final de la aplicación de la condición de contorno.

Con esta modificación, junto con las otras generales, ya se puede ejecutar el runkratos.exe y realizar el cálculo del problema.

Una vez realizado, accediendo al apartado de postproceso de GiD, ya se puede acceder a la visualización de los resultados obtenidos.

5.5.2.2 Imposición de la aceleración horizontal como condición de contorno sísmica

En este caso, el proceso personalizado (*custom process*) que se crea en el archivo *KratosStructuralOpenMP* es el siguiente:

```
######################################################################
# 1 CREATE CUSTOM PROCESSES
import apply_nodal_variable_process
import convert_peer_motion_to_table
import ground_motion_process

acc_record = convert_peer_motion_to_table.convert('H-E12140.AT2')
print('KRATOS: ', acc_record.__class__.__name__)
customProcesses = []
class nodal_variable_settings:
    file_name = "ground_motion_process"
    class_name = "ApplyGroundMotion"
    group_id = 0
    ground_motion_type = "ACCELERATION"
    ground_motion_table = acc_record
    displacement_component = 'X'
    nodes = [1,14]
    user_spatial_function = None
    fix = True
    value = None
    default_value = 0.0
    keep_after_end = False


proc1 = ground_motion_process.CreateCustomProcess(model_part,
nodal_variable_settings)
customProcesses.append(proc1)
######################################################################
```

Como se puede ver:

- Se crea la clase *nodal_variable_settings*, donde se definen todos los parámetros necesarios.

- Se utiliza el proceso *ground_motion_process*. Como se ha explicado en el capítulo 4, este proceso es el encargado de realizar la doble integración a la aceleración con la finalidad de transformarla a desplazamiento. Una vez obtenido, se llama al proceso *apply_nodal_variable_process* y se aplica el desplazamiento correspondiente a cada instante de tiempo.
- Las aceleraciones impuestas como condición de contorno vienen dadas en el archivo 'H-E12140.AT2'. Tampoco son dados en un formato de entrada tipo tabla (tiempo – valor) como es requerido, sino que son dados en forma de lista, espaciados entre ellos y cada dato separado por un tiempo de 0.005s. Se va a requerir el uso de la función *convert_peer_motion_to_table* para obtener una tabla tiempo – desplazamiento.
- La variable de estudio es llamada "ACCELERATION", aplicada en dirección "X".
- Se definen los nodos a los que se les aplicarán estas condiciones de contorno, el 1 y el 14.

5.5.3 Análisis de los resultados obtenidos correspondientes al desplazamiento horizontal

Primeramente, se debe comprobar que la condición de contorno ha sido introducida correctamente. Para ello, se comprueba la evolución del desplazamiento horizontal de los puntos base del pilar, los nodos 1 y 14.

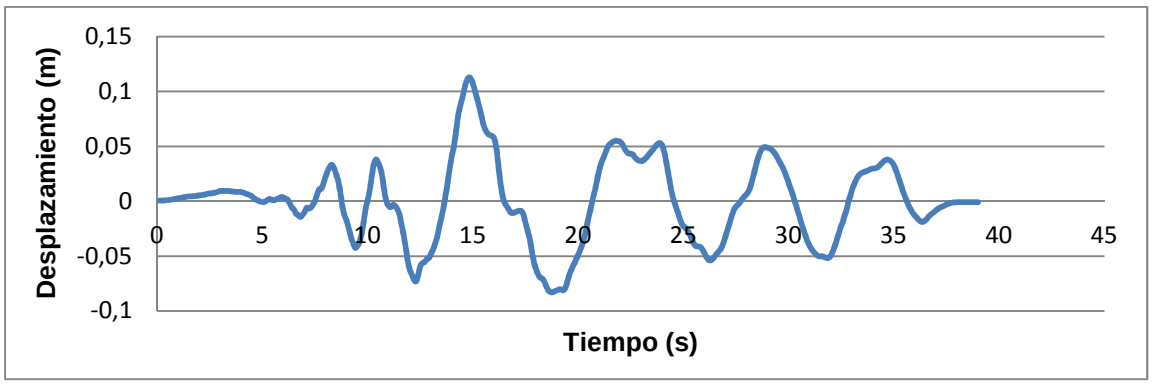

Figura 44. Gráfica correspondiente a la evolución temporal del desplazamiento en los puntos de apoyo del pórtico, nodos 1 y 14.

Se observa que el desplazamiento impuesto es el mismo para los dos nodos, y que es el mismo valor de la tabla procedente del archivo 'H-E12140.DT2'. Por tanto, la condición de contorno ha sido aplicada correctamente.

El desplazamiento global de la estructura, en algunos pasos de tiempo, son los siguientes:

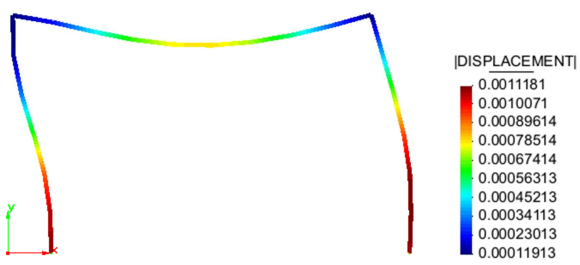

Figura 45. Desplazamiento, paso 10

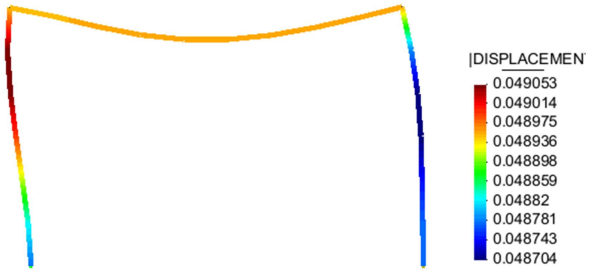

Figura 46. Desplazamiento, paso 20

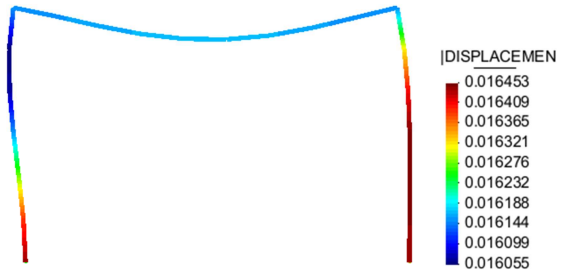

Figura 47. Desplazamiento, paso 30

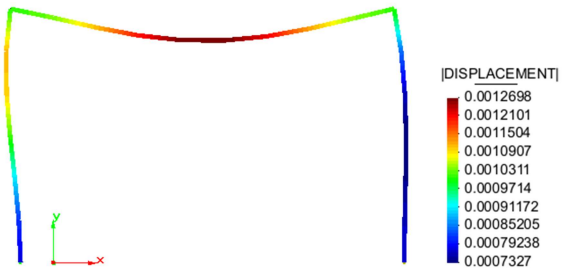

Figura 48. Desplazamiento, paso 37

Una vez comprobada la correcta imposición de las condiciones de contorno, se procede a realizar la comparación de los resultados obtenidos con los del programa OpenSees. El análisis se realiza hasta un tiempo final de 10s, con un paso de tiempo de 0,01s.

El nodo de estudio va a ser el número 22, situado en el extremo superior izquierdo del pórtico. La variación del desplazamiento horizontal en este nodo es la siguiente:

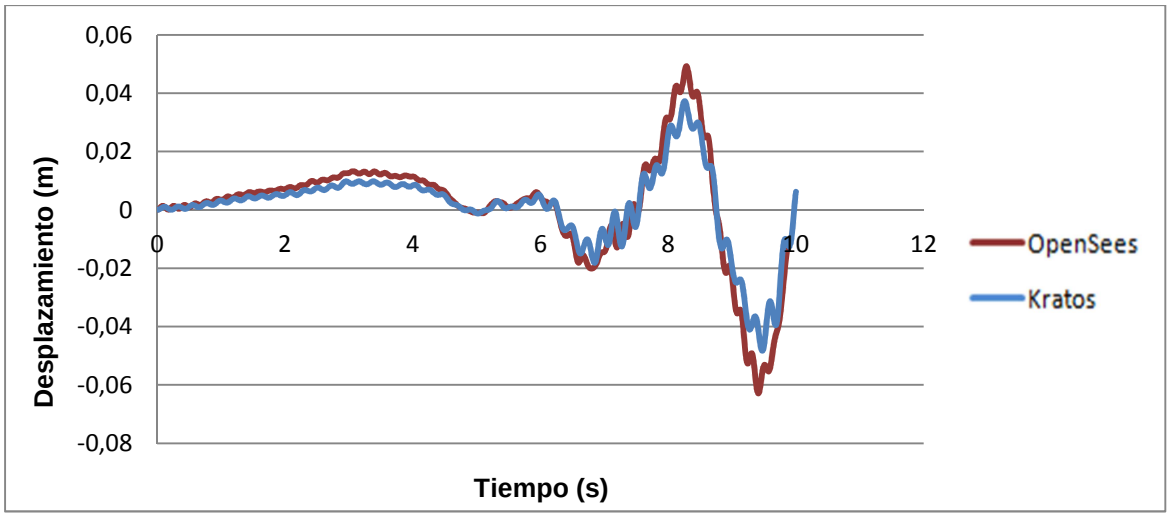

Figura 49. Gráfica correspondiente a la variación del desplazamiento del nodo 22 calculado con Kratos y OpenSees

Comparando los resultados obtenidos mediante los dos programas, se observa que la variación del desplazamiento en el nodo es cualitativamente la misma calculada con los dos. El error relativo es de aproximadamente un 20% de media. Esto es debido a las divergencias en el cálculo entre los dos programas. Por ejemplo, puede ser debido a una diferencia entre las formulaciones empleadas para el cálculo o al uso de otro algoritmo de integración. También puede ser debido a que el modelo de OpenSees tiene sólo tres elementos de mallado y a que la masa del pórtico no es aplicada de forma distribuida como en Kratos, sino que es definida de forma puntual en los nodos.

5.5.4 Análisis de los resultados obtenidos correspondientes a la aceleración horizontal

El desplazamiento horizontal al que se ve expuesto los nodos de las bases de los pilares, 1 y 14, es el siguiente:

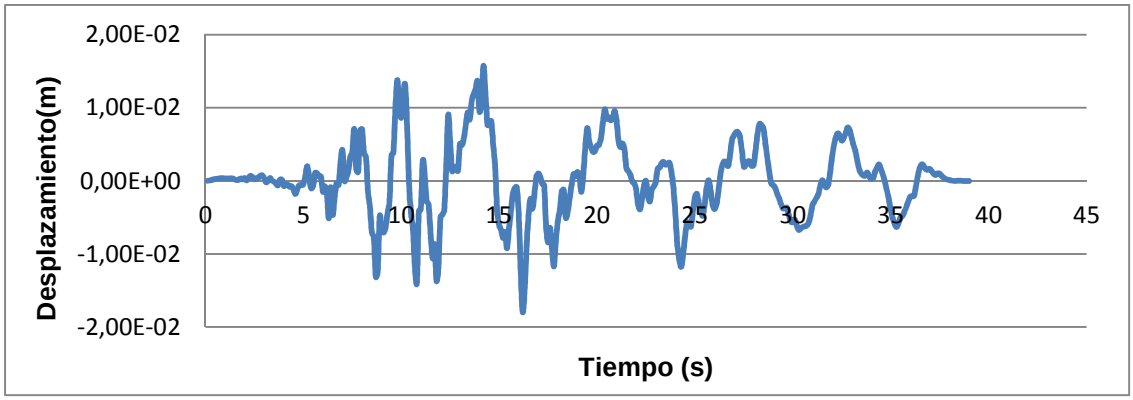

Figura 50. Gráfica correspondiente a la evolución temporal del desplazamiento en los puntos de apoyo del pórtico, nodos 1 y 14.

Como se puede observar, los resultados no son los equivalentes al desplazamiento impuesto directamente como variable vistos en el ejemplo anterior. Esto es debido a que, como se ha enunciado ya en el capítulo 4, la integración se ha realizado mediante el método del trapecio. Este método no da como resultado buenas aproximaciones. Para tratar de realizar un análisis de forma correcta, se debería utilizar otro método que realizara la integración adecuada. La conclusión a la que se debe llegar, fruto de este ejemplo, es que la aceleración ha sido impuesta y aplicada correctamente en los nodos como condición de contorno variable.

Adicionalmente, se debe comentar que en este caso se ha realizado una modificación en la función convert_peer_motion_to_table, debido a que los datos de entrada del archivo 'H-E12140.AT2' vienen dados ya con las unidades del sistema internacional, a diferencia de los del archivo 'H-E12140.DT2', que vienen dados en centímetros. Simplemente se ha modificado la función de forma que estructurara directamente la lista de entrada en formato tabla tiempo – aceleración, sin dividir por 100.

5.6 SISTEMA DE PÓRTICOS SOMETIDO A ACCIÓN SÍSMICA

El ejemplo utilizado a continuación también ha sido seleccionado de un conjunto de ejemplos de OpenSees, como los dos anteriores.

5.6.1 Descripción del problema

La diferencia de este ejemplo con los anteriores es que la estructura de análisis consiste en un sistema de pórticos. Los otros condicionantes son los mismos: compuesto por pilares y vigas, formando un ángulo recto entre ellos y biapoyado. Cada pórtico mide 6m de altura por 9m de ancho, y los pilares y las vigas son de sección rectangular, pero de dimensiones 0.5x0.8m y 0.5x0.5m, respectivamente. El material por el que están formados es el acero, de características:

Módulo de Elasticidad	$2{,}0 \cdot 10^{11}$	N/m^2
Número de Poisson	0,29	-
Densidad	7870	Kg/m^3

Tabla 7. Características mecánicas

Esta estructura va a estar sometida a su peso propio y a un desplazamiento horizontal impuesto en los apoyos y variable en el tiempo, debido a un sismo. Este

desplazamiento horizontal viene proporcionado a partir de una lista de datos de OpenSees. El desplazamiento horizontal impuesto es el que en el caso anterior, obtenido del archivo “H-E12140.DT2”.

El esquema estructural es el siguiente:

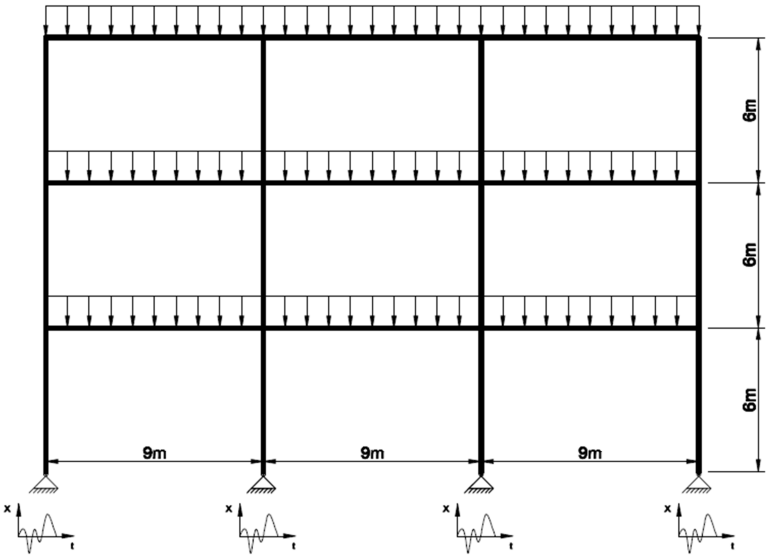

Figura 51. Esquema estructural

El objetivo de este análisis es el de comparar los resultados obtenidos con los del programa OpenSees y comprobar cómo se aplican las condiciones de contorno en un problema un poco más complejo que no los realizados hasta el momento.

5.6.2 Procedimiento a seguir para la resolución del problema

Primero de todo, se introduce la geometría dada en GiD.

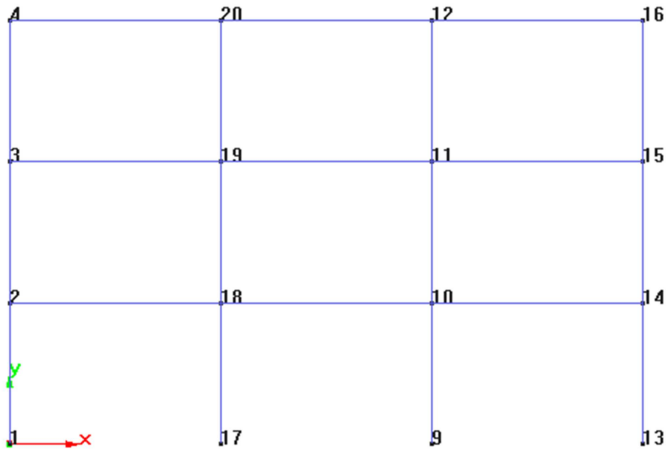

Figura 52. Geometría

Se declara a Kratos como *problemtype*, para así poder aplicar sus condiciones al modelo. Una vez inicializado, se define el material de estudio, con las propiedades mecánicas anteriormente descritas.

Las características del modelo son las mismas que las definidas en el ejemplo anterior.

En este caso se trata de una malla no estructurada triangular, compuesta de 148 nodos y de 153 elementos. La malla se ha realizado de forma más fina debido a que el problema es de mayor tamaño.

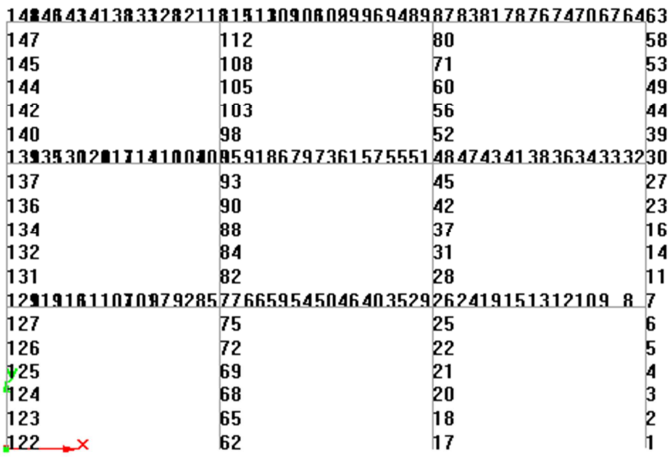

Figura 53. Malla

Ahora sí que se puede realizar el “precálculo”. En la carpeta donde se almacenan los ficheros del problema de estudio veremos que se obtienen los archivos *KratosStructuralOpenMP y ProjectParameters*. Es sobre estos archivos, como se ha explicado en el capítulo 4, donde se podrán realizar las modificaciones convenientes para así poder introducir las cargas variables en el tiempo del problema.

En este caso, el proceso personalizado (*custom process*) que se crea en el archivo *KratosStructuralOpenMP* es el siguiente:

```
######################################################################
# 1 CREATE CUSTOM PROCESSES
import apply_nodal_variable_process
import convert_peer_motion_to_table

disp_record = convert_peer_motion_to_table.convert('H-E12140.DT2')
print('KRATOS: ', disp_record.__class__.__name__)
customProcesses = []
class nodal_variable_settings:
    # these 2 parameters are for the generation of the process
    file_name = "apply_nodal_variable_process"
    class_name = "apply_nodal_variable"
    group_id = 0
```

```
    # parameters used by this custom process
    fix = True
    value = None
    time_table = disp_record,
    time_begin = 0.0
    time_end = 39.0
    default_value = 0.0
    keep_after_end = False
    nodes = [1,17,62,122]
    variable_name = "DISPLACEMENT_X"

proc1 = apply_nodal_variable_process.CreateCustomProcess(model_part,
nodal_variable_settings)
customProcesses.append(proc1)
#######################################################################
```

Como se puede ver:

- Se crea la clase *nodal_variable_settings*, donde se definen todos los parámetros necesarios.
- Se utiliza el proceso *apply_nodal_variable_process* para poder aplicar los desplazamientos en los nodos.
- Los desplazamientos impuestos como condición de contorno vienen dados en el archivo 'H-E12140.DT2', como en el caso anterior.
- La variable de estudio es llamada "DISPLACEMENT_X".
- Se definen los nodos a los que se les aplicarán estas condiciones de contorno, 1, 17, 62 y 122.
- Se establecen el tiempo de inicio y final de la simulación.

Con esta modificación, junto con las otras generales, ya se puede ejecutar el runkratos.exe y realizar el cálculo del problema.

Una vez realizado, accediendo al apartado de postproceso de GiD, ya se pueden ver los resultados obtenidos.

5.6.3 Análisis de los resultados obtenidos

Primeramente, se debe comprobar que la condición de contorno ha sido introducida correctamente. Para ello, se comprueba la evolución del desplazamiento horizontal de los puntos base de los pórticos, los nodos 1,17, 62 y 122.

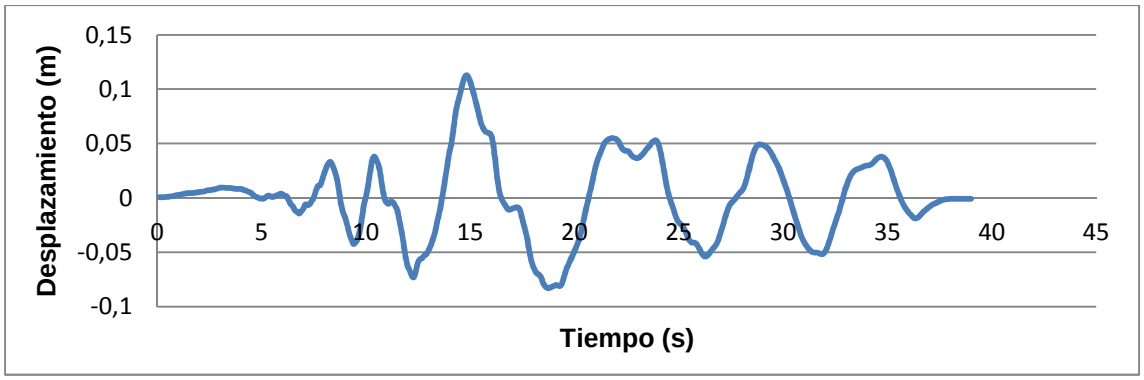

Figura 54. Gráfica correspondiente a la evolución temporal del desplazamiento en los puntos de apoyo del sistema de pórticos

Se observa que el desplazamiento impuesto es el mismo para los dos nodos, y que es el mismo valor de la tabla procedente del archivo 'H-E12140.DT2'; igual que en el ejemplo anterior. Por tanto, la condición de contorno ha sido aplicada correctamente.

A continuación, se representan algunos pasos del desplazamiento horizontal global de toda la estructura:

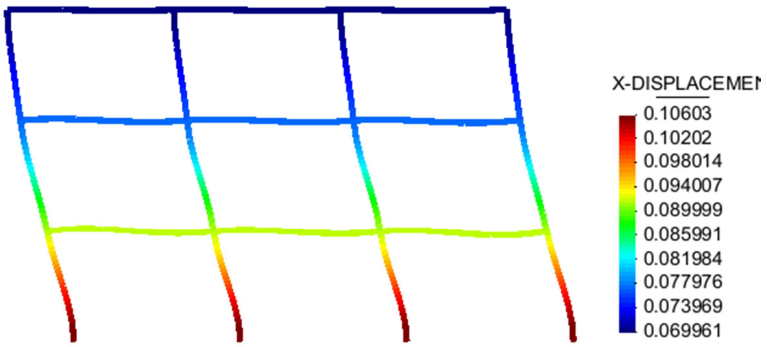

Figura 55. Desplazamiento horizontal, paso 15

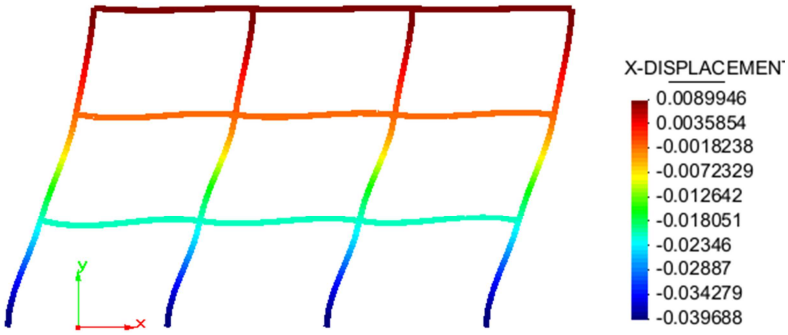

Figura 56. Desplazamiento horizontal, paso 32,21

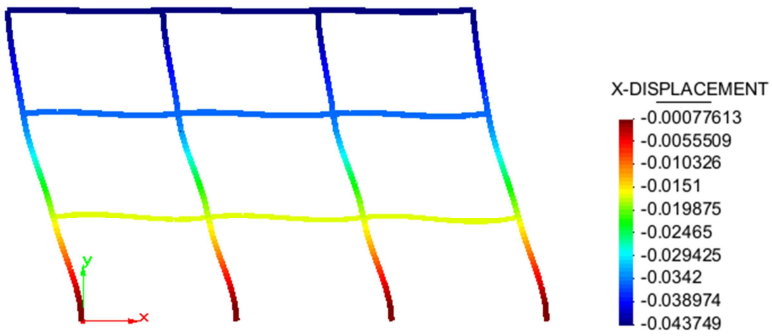

Figura 57. Desplazamiento horizontal, paso 39

Después de comprobar la correcta imposición de las condiciones de contorno, se procede a realizar la comparación de los resultados obtenidos con los del programa OpenSees. El análisis se realiza hasta un tiempo final de 10s, con un paso de tiempo de 0,01s.

El nodo de estudio va a ser el número 148, situado en el extremo superior izquierdo del pórtico.

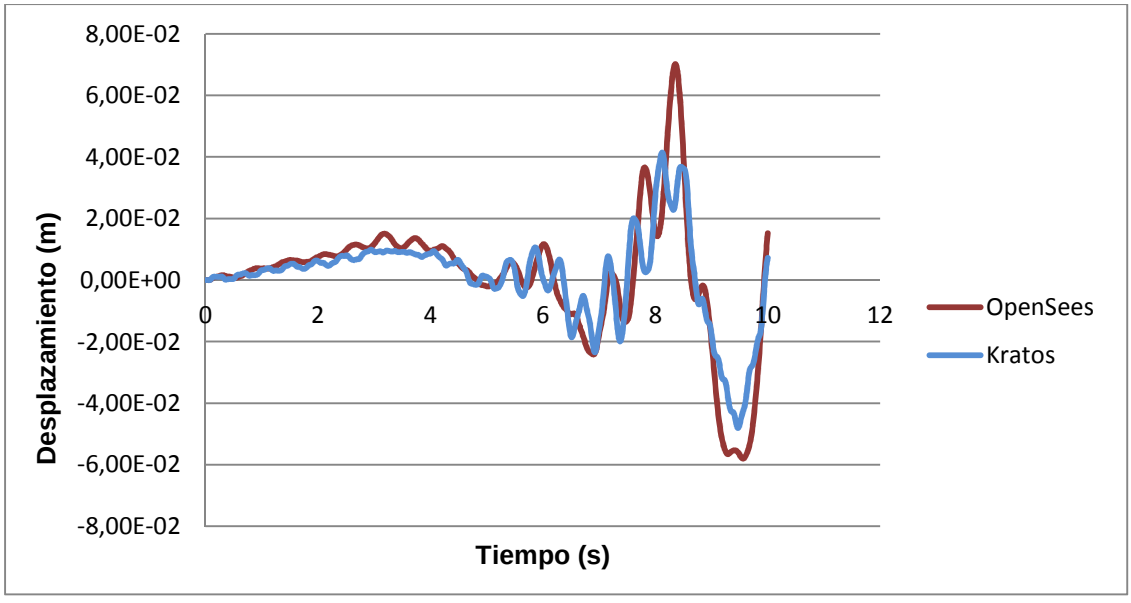

Figura 58. Gráfica correspondiente a la variación del desplazamiento del nodo 148 calculado con Kratos y OpenSees

Al igual que con el ejemplo anterior, la tendencia de las dos gráficas cualitativamente es la misma. La diferencia entre los resultados obtenidos se debe a que los programas de cálculo emprados presentan diferencias en sus procesos de cálculo. También puede se debido a que el modelo de OpenSees tiene sólo tres elementos de mallado y

a que la masa del pórtico no es aplicada de forma distribuida como en Kratos, sino que es definida de forma puntual en los nodos.

CAPÍTULO 6

CONCLUSIONES

El objetivo principal de este trabajo ha sido el de desarrollar y validar una interfaz para Kratos que posibilita la aplicación de condiciones de contorno variables respecto del tiempo. Gracias a este proceso es posible analizar todo tipo de acciones dinámicas, acciones que hasta el momento no eran posibles de analizarse en esta herramienta de cálculo. Una de las características básicas de la interfaz creada es que no se concibió para un problema específico, sino que se ha diseñado como una herramienta que pueda aplicarse a un gran campo de problemas físicos y que, además, permita particularizar el proceso en el caso que sea necesario.

A continuación, se procede a comentar las conclusiones más destacadas de cada capítulo del trabajo.

Los aspectos más relevantes del capítulo 1. Introducción, 2. Estado del arte y 3.Metodología del método de los elementos finitos son:

- En el análisis estructural es determinante el estudio de cualquiera de las acciones, estáticas o dinámicas, a las cuales pueda estar sometida la estructura. Las condiciones de contorno tienen que representar el efecto de dichas acciones.
- Actualmente Kratos no tiene la posibilidad de aplicar este tipo de condiciones, lo que conlleva que no pueden realizarse análisis de problemas sometidos a acciones dinámicas.
- En la etapa de cálculo, existe la posibilidad de implementar una interfaz programada en Python que permita realizar cambios en el proceso convencional de cálculo de Kratos.

Los aspectos más relevantes del capítulo 4. Proceso de imposición de condiciones de contorno variables son:

- La interfaz desarrollada crea un proceso que permite aplicar a los nodos condiciones de contorno variables en el tiempo. Permite aplicarlas independientemente del tipo de variable, de forma que no se focaliza sólo en el análisis estructural. Además, se ofrece la posibilidad de particularizar el proceso realizando modificaciones en el código. Por tanto, se puede decir que

la idea general de este proceso es la misma de Kratos: multifísico y extensible cuando sea necesario.

- Se ha implementado el proceso *apply_nodal_variable_process* que es el encargado de aplicar las condiciones de contorno variables a los nodos correspondientes. Ofrece la posibilidad de utilizar tres distintos formatos de entrada: en forma de una tabla tiempo – valor, definiendo una función respecto del tiempo o a través de un valor constante.
- Se ha implementado el proceso *ground_motion_process* que es una particularización del proceso *apply_nodal_variable_process.* Ofrece la novedad de poder aplicar como variables velocidades y aceleraciones, condiciones de contorno sísmicas típicamente.
- Para implementar la interfaz en el proceso de cálculo, es necesario realizar una serie de modificaciones en los archivos generados de Kratos.

Los aspectos más relevantes del capítulo 5. Validación y ejemplos son los siguientes:

- Los procesos *apply_nodal_variable_process* y *ground_motion_process* realizan la aplicación de las condiciones de contorno variables de forma correcta sobre los nodos correspondientes, independientemente de cuál sea el formato de entrada.
- Los resultados obtenidos comparados con la solución teórica correspondiente han sido los esperados, tanto para análisis de elementos tipo *beam* como de tipo *shell.*
- Las comparaciones realizadas con los programas XFINAS y OpenSees muestran que los resultados, cualitativamente, han sido satisfactorios. Las diferencias observadas, respecto de las calculadas con Kratos, son básicamente debidas a las divergencias en el proceso de cálculo realizado por cada uno.

Se debe resaltar que para la confección de esta tesina el autor ha tenido que realizar un trabajo previo considerable, que ha consistido en el aprendizaje de todas las herramientas necesarias para poder desarrollar la interfaz. Todo este proceso previo ha consistido en:

- La realización de diferentes tutoriales en los que se trataban de generación problemas y visualización de resultados en GiD.
- Varios tutoriales que utilizan Kratos como *problemtype*.

- Aprendizaje del lenguaje de programación Python, juntamente con el análisis de las distintas modificaciones que se pueden implementar utilizándolo.
- Estudio previo de la teoría y metodología del método de los elementos finitos.

Gracias a la realización de este trabajo he podido introducirme en el mundo de los métodos numéricos y de la programación por ordenador, dos temáticas que, a pesar de estos cuatro años del grado, eran prácticamente desconocidas para mí. Se tratan de unas herramientas muy potentes, pero que se necesita un conocimiento profundo de su funcionamiento por parte del usuario para poder utilizarlas.

Todas las habilidades y conocimientos adquiridos van a serme de gran ayuda para la continuación de mis estudios.

Por último, comentar que aunque se hayan cumplido los objetivos principales planteados al inicio de la tesina, se abre una línea de trabajo futuro muy interesante en relación a la validación de la interfaz. En este trabajo se han empezado a realizar algunas, pero aún pueden comprobarse la convergencia de los resultados obtenidos de gran variedad de problemas con los resultados teóricos y sobre todo con los de otros programas que también utilicen el método de los elementos finitos.

CAPÍTULO 7

AGRADECIMIENTOS

Primer de tot, m'agradaria donar les gràcies a la meva família, amics i companys de classe per donar-me la força i l'empenta necessària, no només per realitzar aquest treball, sinó per tots aquests quatre anys de durada del grau. Especialment ho agraeixo a l'Aina i a l'Assís, sense ells no ho hagués aconseguit.

Además, quisiera mostrar mi agradecimiento a mis tutores: Antonia, Pooyan y Massimo. Han tenido mucha paciencia conmigo y siempre han estado accesibles a recibir mis consultas y resolver mis dudas.

CAPÍTULO 8

REFERENCIAS

[1] Kratos, Multiphysics finite element method C++ open source code.
http://kratos-wiki.cimne.upc.edu/

[2] CIMNE, Centro Internacional de Métodos Numéricos en Ingeniería.
http://www.cimne.com/cdl1/spacehome/2/0#

[3] Python, programming language.
https://www.python.org/

[4] GiD, the personal pre and post processor.
http://www.gidhome.com/

[5] GiD Reference Manual.

[6] Oñate Ibañez de Navarra, Eugenio. “Cálculo de Estructuras por el Método de Elementos Finitos. Análisis estático lineal”, 1995, ISBN: 84-87867-00-6

[7] Barbat, Alex H; Canet, Joan Miquel. “Estructuras sometidas a acciones sísmicas”, 1994, ISBN: 84-87867-10-3.

[8] Manual de ejemplos de OpenSees.
http://opensees.berkeley.edu/wiki/index.php/Examples_Manual

[9] XFINAS, Nonlinear Dynamic Analysis Software.
http://www.xfinas.com/

[10] XFINAS Validation Manual.

APÉNDICE

AP.1 Proceso Apply_nodal_variable_process

```
from KratosMultiphysics import *

from KratosMultiphysics.SolidMechanicsApplication  import*

# This process
class ApplyNodalVariable:
    def __init__(self,
                 model,
                 variable_name,
                 time_begin,
                 time_end,
                 value=None,
                 time_table = None,
                 nodes=[],
                 fix=False,
                 user_time_function=None,
                 user_spatial_function=None,
                 default_value = 0.0,
                 keep_after_end=False,
                 ):
        self.myModel = model
        self.value = value
        self.time_table = time_table
        self.nodes = nodes
        self.fix = fix
        self.variable_name = variable_name
        self.time_begin = time_begin
        self.time_end = time_end
        self.user_time_function = user_time_function
        self.user_spatial_function = user_spatial_function
        self.default_value = default_value
        self.keep_after_end = keep_after_end

    # Execute method is used to execute the Process algorithms.
    def Execute(self):
        pass

    # this function is designed for being called at the beginning of
the computations
    # right after reading the model and the groups
    def ExecuteInitialize(self):
        pass

    # this function is designed for being execute once before the
solution loop but after all of the
    # solvers where built
    def ExecuteBeforeSolutionLoop(self):
        pass

    # this function will be executed at every time step BEFORE
performing the solve phase
    def ExecuteInitializeSolutionStep(self):
```

```
        current_time = self.myModel.ProcessInfo[TIME]
        theVariable = globals()[self.variable_name] # target variable

        if(current_time >= self.time_begin):
            if(current_time <= self.time_end):

                # 1. find the current value (constant or table or
function of time)
                current_value = 0.0
                if(self.value is not None):
                    current_value = self.value
                elif(self.time_table is not None):
                    current_value =
self.time_table.get_factor(current_time)
                elif(self.user_time_function is not None):
                    current_value =
self.user_time_function(current_time)

                else:
                    raise Exception("you need to specify something")

                # loop over nodes...
                has_spatial_function = (self.user_spatial_function is
not None)
                for node_id in self.nodes:
                    iNode = self.myModel.Nodes[node_id]
                    current_nodal_value = current_value
                    # 2. interpolate in space if necessary
                    if(has_spatial_function):
                        x = iNode.X
                        y = iNode.Y
                        z = iNode.Z
                        current_nodal_value =
self.user_spatial_function(x,y,z,current_value)
                    # 3. set the value
                    iNode.SetSolutionStepValue(theVariable,
current_nodal_value)
                    if(self.fix):
                        iNode.Fix(theVariable)
            else:
                if(self.keep_after_end == False):
                    current_value = self.default_value
                    for node_id in self.nodes:
                        iNode = self.myModel.Nodes[node_id]
                        iNode.SetSolutionStepValue(theVariable,
current_value)
                        iNode.Free(theVariable)

    # this function will be executed at every time step AFTER
performing the solve phase
    def ExecuteFinalizeSolutionStep(self):
        pass

    # this function will be executed at every time step BEFORE
writing the output
    def ExecuteBeforeOutputStep(self):
        pass
```

```
    # this function will be executed at every time step AFTER writing
the output
    def ExecuteAfterOutputStep(self):
        pass

    # this function is designed for being called at the end of the
computations
    # right after reading the model and the groups
    def ExecuteFinalize(self):
        pass

# Factory function:
# This should be the same for all processes
def CreateCustomProcess(model, settings):
    # you have to do this for all MANDATORY attributes
    if(hasattr(settings, "variable_name")):
        variable_name = settings.variable_name
    else:
        raise Exception("ERROR: attribute 'variable_name' is not
defined. It's a MANDATORY variable")

    if(hasattr(settings, "time_begin")):
        time_begin = settings.time_begin
    else:
        raise Exception("ERROR: attribute 'time_begin' is not defined.
It's a MANDATORY variable")

    if(hasattr(settings, "time_end")):
        time_end = settings.time_end
    else:
        raise Exception("ERROR: attribute 'time_end' is not defined.
It's a MANDATORY variable")

    if(hasattr(settings,"nodes")):
        nodes = settings.nodes
    else:
        raise Exception("ERROR: attribute 'nodes' is not defined. It's
a MANDATORY variable")

    # optional values
    value = None
    fix = False
    user_time_function = None
    user_spatial_function = None
    default_value = 0.0
    keep_after_end = False
    time_table = None
    if(hasattr(settings, "value")):
        value = settings.value
    if(hasattr(settings,"fix")):
        fix = settings.fix
    if(hasattr(settings,"time_table")):
        time_table = settings.time_table
        if time_table is not None:
            time_table = InterpolatedTimeTable(time_table)
    if(hasattr(settings,"user_time_function")):
        user_time_function = settings.user_time_function
    if(hasattr(settings,"user_spatial_function")):
        user_spatial_function = settings.user_spatial_function
    if(hasattr(settings,"default_value")):
```

```
        default_value = settings.default_value
    if(hasattr(settings,"keep_after_end")):
        keep_after_end = settings.keep_after_end

    # construct the actual process
    return ApplyNodalVariable(model,
                              variable_name,
                              time_begin,
                              time_end,
                              value,
                              time_table,
                              nodes,
                              fix,
                              user_time_function,
                              user_spatial_function,
                              default_value,
                              keep_after_end)


class InterpolatedTimeTable:
    def __init__(self, table=[]):
        self.table = table
    def get_factor(self, current_time):
        last_time = None
        last_value = None
        current_value = None
        for ir in self.table:
            ir_time = ir[0]
            ir_value = ir[1]
            if(last_time is not None):
                delta_time = ir_time-last_time
                tolerance = delta_time * 1.0E-10
                if(current_time <= ir_time+tolerance):
                    factor = (current_time-last_time)/delta_time
                    current_value = last_value + factor*(ir_value-
last_value)
                    break
            last_time = ir_time
            last_value = ir_value
        return current_value
    def get_duration(self):
        n = len(self.table)
        tuple_0 = self.table[0]
        tuple_end = self.table[n-1]
        return tuple_end[0]-tuple_0[0]
    def append(self, time, value):
        self.table.append( (time,value) )
    def get_initial_time(self):
        init_time = 0.0
        if(len(self.table) > 0):
            tuple_0 = self.table[0]
            init_time = tuple_0[0]
        return init_time
    def get_final_time(self):
        final_time = 0.0
        n = len(self.table)
        if(n > 0):
            tuple_end = self.table[n-1]
            final_time = tuple_end[0]
        return final_time
```

```
    def get_number_of_rows(self):
        return len(self.table)
    def get_delta_time(self):
        return self.table[1][0]-self.table[0][0]
    def __str__(self):
        s = ''
        for i in self.table:
            s = s+('{},\t{}\n'.format(i[0], i[1]))
        return s
```

AP.2 Proceso Ground_motion_process

```
from KratosMultiphysics import *
import apply_nodal_variable_process


def IntegrateSeismicRecord(input_table, delta_time):

    duration = input_table.get_duration()
    num_steps = int(duration/delta_time) + 1
    output_table =
apply_nodal_variable_process.InterpolatedTimeTable() # Empty table

    # first create the initial tuple
    initial_time = input_table.get_initial_time()
    initial_value = input_table.get_factor(initial_time)
    initial_value_output = initial_value*delta_time*0.5
    output_table.append(initial_time, initial_value_output)

    previous_time = initial_time
    previous_value = initial_value
    previous_value_output = initial_value_output
    current_time = initial_time

    for i in range(1, num_steps):
        # update the current state
        current_time += delta_time
        current_value = input_table.get_factor(current_time)

        # integrate
        integrated_value = previous_value_output +
delta_time*0.5*(current_value+previous_value)

        # update of previous values
        previous_value = current_value
        previous_value_output = integrated_value

        # append to the output table
        output_table.append(current_time, integrated_value)

    return output_table

def CreateCustomProcess(model, settings):
    if(hasattr(settings, "ground_motion_type")):
        ground_motion_type = settings.ground_motion_type
```

```
    else:
        raise Exception("ERROR: attribute 'ground_motion_type' is not
defined. It's a MANDATORY variable")

    if(hasattr(settings, "ground_motion_table")):
        ground_motion_table =
apply_nodal_variable_process.InterpolatedTimeTable(settings.ground_mot
ion_table)
    else:
        raise Exception("ERROR: attribute 'ground_motion_table' is not
defined. It's a MANDATORY variable")

    if(hasattr(settings, "displacement_component")):
        displacement_component = settings.displacement_component
    else:
        raise Exception("ERROR: attribute 'displacement_component' is
not defined. It's a MANDATORY variable")

    if(hasattr(settings, "nodes")):
        nodes = settings.nodes
    else:
        raise Exception("ERROR: attribute 'nodes' is not defined. It's
a MANDATORY variable")

    # optional values
    user_spatial_function = None
    default_value = 0.0
    keep_after_end = False

    if(hasattr(settings,"user_spatial_function")):
        user_spatial_function = settings.user_spatial_function
    if(hasattr(settings,"default_value")):
        default_value = settings.default_value
    if(hasattr(settings,"keep_after_end")):
        keep_after_end = settings.keep_after_end

    # construct the actual process
    return ApplyGroundMotion(ground_motion_type,
                             ground_motion_table,
                             model,
                             displacement_component,
                             nodes,
                             user_spatial_function,
                             default_value,
                             keep_after_end)

class ApplyGroundMotion:
    # same constructor as ApplyNodalValue
    def __init__(self,
                 ground_motion_type, # string ['DISPLACEMENT',
'VELOCITY', 'ACCELERATION']
                 ground_motion_table,
#apply_nodal_variable_process.InterpolatedTimeTable
                 model,
                 displacement_component, # string ['X', 'Y', 'Z']
                 nodes,
                 user_spatial_function,
                 default_value,
                 keep_after_end
                 ):
```

```
        self.ground_motion_type = ground_motion_type

        if(self.ground_motion_type == "VELOCITY"):
            # integrate once
            delta_time = ground_motion_table.get_duration() /
(ground_motion_table.get_number_of_rows()-1)
            self.ground_motion_table =
IntegrateSeismicRecord(ground_motion_table, delta_time)
        elif(self.ground_motion_type == "ACCELERATION"):
            #integrate twice
            delta_time = ground_motion_table.get_duration() /
(ground_motion_table.get_number_of_rows()-1)
            velocity_table =
IntegrateSeismicRecord(ground_motion_table, delta_time)
            delta_time = velocity_table.get_duration() /
(velocity_table.get_number_of_rows()-1)
            self.ground_motion_table =
IntegrateSeismicRecord(velocity_table, delta_time)
        else:
            #this should be DISPLACEMENT
            if(self.ground_motion_type == "DISPLACEMENT"):
                # no need to integrate
                self.ground_motion_table = ground_motion_table
            else:
                raise Exception("Unknown Ground motion type")

        variable_name = "DISPLACEMENT_" + displacement_component

        self.my_inner_process =
apply_nodal_variable_process.ApplyNodalVariable (model,
                                                  variable_name,

self.ground_motion_table.get_initial_time(),

self.ground_motion_table.get_final_time(),
                                                  None,

self.ground_motion_table,
                                                  nodes,
                                                  True,
                                                  None,

user_spatial_function,
                                                  default_value,
                                                  keep_after_end)

    def Execute(self):
        self.my_inner_process.Execute()

    def ExecuteInitialize(self):
        self.my_inner_process.ExecuteInitialize()

    def ExecuteBeforeSolutionLoop(self):
        self.my_inner_process.ExecuteBeforeSolutionLoop()

    def ExecuteInitializeSolutionStep(self):
        self.my_inner_process.ExecuteInitializeSolutionStep()

    def ExecuteFinalizeSolutionStep(self):
        self.my_inner_process.ExecuteFinalizeSolutionStep()
```

```
    def ExecuteBeforeOutputStep(self):
        self.my_inner_process.ExecuteBeforeOutputStep()

    def ExecuteAfterOutputStep(self):
        self.my_inner_process.ExecuteAfterOutputStep()

    def ExecuteFinalize(self):
        self.my_inner_process.ExecuteFinalize()
```

AP. 2.1 Función Convert_peer_motion_to_table

```
import os
import glob
import re

def convert(filename):

    lines = [line.strip() for line in open(filename)]

    L0 = lines[3].split(',')
    num_points = int(L0[0].split('=')[1].strip())
    dt = float(re.sub('[SEC]','',
L0[1].split('=')[1].strip()).strip())

    table = []

    current_time = 0.0
    table.append((current_time,0.0))
    for i in range(4,len(lines)):
        values=lines[i].split()
        for ival in values:
            current_time += dt
            table.append((current_time, float(ival.strip())/100.0))

    return table
```

AP.3 KratosStructuralOpenMP particularizado para el ejemplo 5.5.1

```
from __future__ import print_function, absolute_import, division
#makes KratosMultiphysics backward compatible with python 2.6 and 2.7
# Activate it to import in the gdb path:
# import sys
# sys.path.append('/home/jmaria/kratos')
# x = raw_input("stopped to allow debug: set breakpoints and press
enter to continue");

#
# ***************GENERAL MAIN OF THE ANALISYS****************###
#
# time control starts
from time import *
print(ctime())
```

```
# measure process time
t0p = clock()
# measure wall time
# t0w = time()

# ----------------------------------------------------------------#
# --CONFIGURATIONS START--####################
# Import the general variables read from the GiD
import ProjectParameters as general_variables

# setting the domain size for the problem to be solved
domain_size = general_variables.domain_size

# including kratos path
from KratosMultiphysics import *

# including Applications paths
from KratosMultiphysics.ExternalSolversApplication import *
from KratosMultiphysics.SolidMechanicsApplication import *

# import the python utilities:
import restart_utility as restart_utils
import gid_output_utility as gid_utils

import conditions_python_utility_mod as condition_utils
import list_files_python_utility as files_utils

import time_operation_utility as operation_utils

# ------------------------#--FUNCTIONS START--#------------------#
# ----------------------------------------------------------------#
# --TIME MONITORING START--###################
def StartTimeMeasuring():
    # measure process time
    time_ip = clock()
    return time_ip


def StopTimeMeasuring(time_ip, process):
    # measure process time
    time_fp = clock()
    print(" ", process, " [ spent time = ", time_fp - time_ip, "] ")
# --TIME MONITORING END --####################

# --SET NUMBER OF THREADS --##################


def SetParallelSize(num_threads):
    parallel = OpenMPUtils()
    print("Num Threads = ", num_threads)
    parallel.SetNumThreads(int(num_threads))
# --SET NUMBER OF THREADS --##################

# ------------------------#--FUNCTIONS END--#--------------------#
# ----------------------------------------------------------------#


# defining the number of threads:
num_threads = general_variables.NumberofThreads
SetParallelSize(num_threads)
```

```
# defining the type, the name and the path of the problem:
problem_type = general_variables.ProblemType
problem_name = general_variables.problem_name
problem_path = general_variables.problem_path

# defining a model part
model_part = ModelPart("SolidDomain")

# defining the model size to scale
length_scale = 1.0

# --DEFINE MAIN SOLVER START--################

SolverSettings = general_variables.SolverSettings

# import solver file
solver_constructor = __import__(SolverSettings.solver_type)

# construct the solver
main_step_solver = solver_constructor.CreateSolver(model_part,
SolverSettings)

# --DEFINE MAIN SOLVER END--##################


# --READ AND SET MODEL FILES--###############

# set the restart of the problem
restart_step = general_variables.Restart_Step
problem_restart = restart_utils.RestartUtility(model_part,
problem_path, problem_name)

# set the results file list of the problem (managed by the
problem_restart and gid_print)
print_lists = general_variables.PrintLists
output_mode = general_variables.GidOutputConfiguration.GiDPostMode
list_files = files_utils.ListFilesUtility(problem_path, problem_name,
print_lists, output_mode)
list_files.Initialize(general_variables.file_list)

# --READ AND SET MODEL FILES END--############


# --DEFINE CONDITIONS START--#################
incr_disp = general_variables.Incremental_Displacement
incr_load = general_variables.Incremental_Load
rotation_dofs = SolverSettings.RotationDofs
conditions = condition_utils.ConditionsUtility(model_part,
domain_size, incr_disp, incr_load, rotation_dofs)

# --DEFINE CONDITIONS END--###################


# --GID OUTPUT OPTIONS START--###############
# set gid print options
gid_print = gid_utils.GidOutputUtility(problem_name,
general_variables.GidOutputConfiguration)

# --GID OUTPUT OPTIONS END--##################
```

```
# --CONFIGURATIONS END--######################
# ------------------------------------------------------------------#


# --START SOLUTION--######################
#
# initialize problem : load restart or initial start
load_restart = general_variables.LoadRestart
save_restart = general_variables.SaveRestart

# set buffer size
buffer_size = 3

# define problem variables:
solver_constructor.AddVariables(model_part, SolverSettings)


# --- READ MODEL ------#
if(load_restart == False):

    # remove results, restart, graph and list previous files
    problem_restart.CleanPreviousFiles()
    list_files.RemoveListFiles()

    # reading the model
    model_part_io = ModelPartIO(problem_name)
    model_part_io.ReadModelPart(model_part)

    # set the buffer size
    model_part.SetBufferSize(buffer_size)
    # Note: the buffer size should be set once the mesh is read for
the first time

    # set the degrees of freedom
    solver_constructor.AddDofs(model_part, SolverSettings)

    # set the constitutive law
    import constitutive_law_python_utility as constitutive_law_utils

    constitutive_law =
constitutive_law_utils.ConstitutiveLawUtility(model_part,
domain_size);
    constitutive_law.Initialize();

else:

    # reading the model from the restart file
    problem_restart.Load(restart_step);

    # remove results, restart, graph and list posterior files
    problem_restart.CleanPosteriorFiles(restart_step)
    list_files.ReBuildListFiles()

# set mesh searches and modeler
# modeler.InitializeDomains();

# if(load_restart == False):
    # find nodal h
```

```
    # modeler.SearchNodalH();


# --- PRINT CONTROL ---#
print(model_part)
print(model_part.Properties[1])


# --INITIALIZE--############################
######################################################################
# 1 CREATE CUSTOM PROCESSES
import apply_nodal_variable_process
import convert_peer_motion_to_table

disp_record = convert_peer_motion_to_table.convert('H-E12140.DT2')
print('KRATOS: ', disp_record.__class__.__name__)
customProcesses = []
class nodal_variable_settings:
    # these 2 parameters are for the generation of the process
    file_name = "apply_nodal_variable_process"
    class_name = "apply_nodal_variable"
    group_id = 0
    # parameters used by this custom process
    fix = True
    value = None
    time_table = disp_record,
    time_begin = 0.0
    time_end = 39.0
    default_value = 0.0
    keep_after_end = False
    nodes = [1,14]
    variable_name = "DISPLACEMENT_X"

proc1 = apply_nodal_variable_process.CreateCustomProcess(model_part,
nodal_variable_settings)
customProcesses.append(proc1)
######################################################################

# set delta time in process info
model_part.ProcessInfo[DELTA_TIME] = general_variables.time_step

# solver initialize
main_step_solver.Initialize()
main_step_solver.SetRestart(load_restart) #calls strategy initialize
if no restart

# initial contact search
# modeler.InitialContactSearch()

#define time steps and loop range of steps
time_step = model_part.ProcessInfo[DELTA_TIME]

# define time steps and loop range of steps
if(load_restart):

    buffer_size = 0

else:

    model_part.ProcessInfo[TIME]                = 0
```

```
    model_part.ProcessInfo[TIME_STEPS]          = 0
    model_part.ProcessInfo[PREVIOUS_DELTA_TIME] = time_step;

    conditions.Initialize(time_step);


# initialize step operations
starting_step  = model_part.ProcessInfo[TIME_STEPS]
starting_time  = model_part.ProcessInfo[TIME]
ending_step    = general_variables.nsteps
ending_time    = general_variables.nsteps * time_step


output_print = operation_utils.TimeOperationUtility()
gid_time_frequency = general_variables.GiDWriteFrequency
output_print.InitializeTime(starting_time, ending_time, time_step,
gid_time_frequency)

restart_print = operation_utils.TimeOperationUtility()
restart_time_frequency = general_variables.RestartFrequency
restart_print.InitializeTime(starting_time, ending_time, time_step,
restart_time_frequency)


# --TIME INTEGRATION--#######################

# writing a single file
gid_print.initialize_results(model_part)

#initialize time integration variables
current_time = starting_time
current_step = starting_step

# filling the buffer
for step in range(0,buffer_size):

  model_part.CloneTimeStep(current_time)
  model_part.ProcessInfo[DELTA_TIME] = time_step
  model_part.ProcessInfo[TIME_STEPS] = step-buffer_size

# writing a initial state results file
current_id = 0
gid_print.write_results(model_part, general_variables.nodal_results,
general_variables.gauss_points_results, current_time, current_step,
current_id)
list_files.PrintListFiles(current_id);

# solving the problem
while(current_time < ending_time):

  # store previous time step
  model_part.ProcessInfo[PREVIOUS_DELTA_TIME] = time_step
  # set new time step ( it can change when solve is called )
  time_step = model_part.ProcessInfo[DELTA_TIME]

  current_time = current_time + time_step
  current_step = current_step + 1
  model_part.CloneTimeStep(current_time)
  model_part.ProcessInfo[TIME] = current_time
```

```
  print("STEP = ", current_step)
  print("TIME = ", current_time)

  ##################################################################
  # 2 INITIALIZE CUSTOM PROCESSES
  for iProc in customProcesses:
    iProc.ExecuteInitializeSolutionStep()
  ##################################################################

  clock_time = StartTimeMeasuring();
  # solve time step non-linear system
  main_step_solver.Solve()
  StopTimeMeasuring(clock_time, "Solving");

  ##################################################################
  # 3 FINALIZE CUSTOM PROCESSES
  for iProc in customProcesses:
    iProc.ExecuteFinalizeSolutionStep()
  ##################################################################

  # incremental load
  conditions.SetIncrementalLoad(current_step, time_step);

  # print the results at the end of the step
  execute_write = output_print.perform_time_operation(current_time)
  if(execute_write):
      clock_time = StartTimeMeasuring();
      current_id = output_print.operation_id()
      # print gid output file
      gid_print.write_results(model_part,
general_variables.nodal_results,
general_variables.gauss_points_results, current_time, current_step,
current_id)
      # print on list files
      list_files.PrintListFiles(current_id);
      StopTimeMeasuring(clock_time, "Write Results");

  # print restart file
  if(save_restart):
      execute_save =
restart_print.perform_time_operation(current_time)
      if(execute_save):
          clock_time = StartTimeMeasuring();
          current_id = output_print.operation_id()
          problem_restart.Save(current_time, current_step,
current_id);
          StopTimeMeasuring(clock_time, "Restart");


  conditions.RestartImposedDisp()

# --FINALIZE--##############################
#

# writing a single file
gid_print.finalize_results()

print("Analysis Finalized ")

# --END--##################################
```

```
#

# measure process time
tfp = clock()
# measure wall time
# tfw = time()

print(ctime())
# print "Analysis Completed  [Process Time = ", tfp - t0p, "seconds,
Wall Time = ", tfw - t0w, " ]"
print("Analysis Completed  [Process Time = ", tfp - t0p, "] ")

# to create a benchmark: add standard benchmark files and decomment
next two lines
# rename the file to: run_test.py
#from run_test_benchmark_results import *
#WriteBenchmarkResults(model_part)
```